Geometric Algorithms for Trajectory Analysis

Frank Staals

This thesis was typeset using Emacs and LuaLaTeX. The main font is TeX Gyre Pagella. For the mathematical expressions, a combination of Tex Gyre Pagella Math, Latin Modern Math, XITTS Math, and Asana Math was used. The sans serif font in section headings, paragraphs, etc. is Latin Modern Sans. The font in the page headers is Iwona. Finally, the font in the title, part, and chapter headings is Archisto.

Almost all figures were drawn in Ipe, the extensible drawing editor. For Figures 1.2, 3.14, and 5.2 a combination of Ipe and Blender was used.

Cover design: Frank Staals. The cover features 3D models of the classic japanese origami crane, and Hideo Komatsu's origami sheep. All models were drawn and rendered using Blender.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

ISBN: 978-90-393-6349-2

Geometric Algorithms for Trajectory Analysis

Geometrische Algoritmen voor Trajectory-analyse

(met een samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Utrecht
op gezag van de rector magnificus, prof.dr. G.J. van der Zwaan,
ingevolge het besluit van het college voor promoties

in het openbaar te verdedigen op
maandag 29 juni 2015 des middags te 12.45 uur

door

Frank Staals

geboren op 10 augustus 1988 te Eindhoven

Promotor: Prof.dr. M.J. van Kreveld
Copromotor: Dr. M. Löffler

This work was supported by the Netherlands Organisation for Scientific Research (NWO) under project number 612.001.022 (*Robust Median Trajectories*).

Contents

I Introduction 1

1 Introduction 3
- 1.1 Overview of Analysis Tasks . . . 4
- 1.2 A Trajectory . . . 7
- 1.3 Outline . . . 9

2 Preliminaries 21
- 2.1 Parametric Search . . . 21

II Analysis Tasks for a Single Trajectory 23

3 Non-Monotone Segmentation 25
- 3.1 General Approach . . . 26
  - 3.1.1 The start-stop diagram . . . 27
- 3.2 Solving the Segmentation Problem . . . 29
  - 3.2.1 Verticonvex cells only . . . 30
  - 3.2.2 At most one tunnel per row . . . 32
- 3.3 Combining Criteria . . . 34
- 3.4 Lower bounds . . . 36
- 3.5 Non-Monotone Segmentation in the Discrete Trajectory Model . 42
- 3.6 Computing the Start-stop Diagram . . . 43
  - 3.6.1 Segmentation on the Outlier-Tolerant Criterion . . . 43
  - 3.6.2 Segmentation on the Standard-Deviation Criterion . . . 49
- 3.7 Line Simplification . . . 52
- 3.8 Concluding Remarks . . . 55

4 Finding Hotspots 57
- 4.1 Properties of Ξ . . . 59
- 4.2 Total Length . . . 60
  - 4.2.1 Fixed Radius . . . 60

4.2.2 Fixed Length 60
4.3 Contiguous Length 62
4.3.1 Fixed Radius 63
4.3.2 Fixed Length 65
4.4 Relative Length 67
4.4.1 Total Length 67
4.4.2 Contiguous Length 73
4.5 Extensions 75
4.6 Concluding Remarks 76

III Analysis Tasks for Multiple Trajectories 79

5 The Trajectory Grouping Structure 81
5.1 Representing the Grouping Structure 83
5.1.1 The Reeb Graph 84
5.1.2 Bounding the Number of Maximal Groups 86
5.2 Computing the Grouping Structure 90
5.2.1 Computing the Reeb Graph 90
5.2.2 Computing the Maximal Groups 91
5.3 Robustness 94
5.3.1 Computation of Maximal Robust Groups 96
5.4 Evaluation 99
5.5 Concluding Remarks 101

6 Grouping under Geodesic Distance 103
6.1 Preliminaries 105
6.2 Simple Polygon 106
6.2.1 Lower Bound 106
6.2.2 Upper Bound 106
6.2.3 Algorithm 108
6.3 Well-spaced Obstacles 110
6.3.1 Lower Bound 110
6.3.2 Upper Bound 111
6.3.3 Algorithm 114
6.4 General Obstacles 114
6.4.1 Lower Bound 114
6.4.2 Upper Bound 116
6.4.3 Algorithm 123
6.5 Concluding Remarks 123

7 Central Trajectories 125
7.1 Entities moving in $\mathbb{R}^1$ 127
7.1.1 Complexity 129
7.1.2 Algorithm 133
7.2 Entities moving in $\mathbb{R}^d$ 135
7.2.1 Complexity 135
7.2.2 Algorithm 138
7.3 Extensions 141
7.4 Concluding Remarks 143

IV Concluding Remarks 145

8 Concluding Remarks 147
8.1 Outlook 148

Samenvatting 153

Bibliography 155

Curriculum Vitae 165

Acknowledgments 167

Part I

Introduction

Chapter 1

Introduction

Movement is a phenomenon that we encounter every day. Almost everyone and everything in our world moves: we walk to go shopping, we cycle to work, we drive to the gym, and we fly to a far-away country for our vacation. It is not just people that move; animals also run, swim, or fly, and even glaciers and hurricanes move. Modern technology, most notably the Global Positioning System (GPS), has made tracking such movement easy and cheap. Moreover, the cost of storing the resulting tracking data –the *trajectories*– has become negligible. Because of these developments trajectory data is ubiquitous today.

Researchers in various fields, ranging from animal ecology [24, 32, 66] and meteorology [108] to traffic control [87] and sports analysis [56], analyze trajectory data. However, unlike collecting and storing movement data, analyzing movement data is not an easy process. Consider for example Figure 1.1(a), which shows the trajectories of five laps on a mountain bike track. Even a

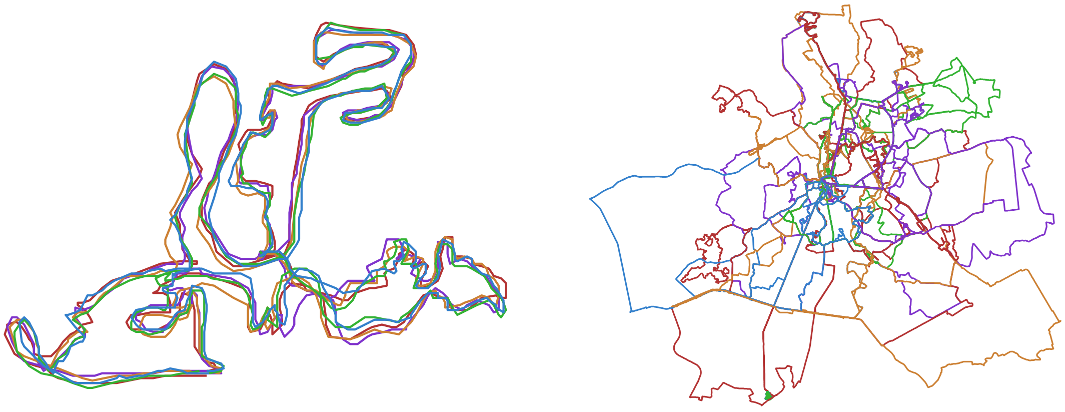

(a) (b)

Figure 1.1: (a) The trajectories of five laps on a mountainbike track. (b) Roughly 100 cycling trajectories.

simple question like “Which two laps are most similar?” is difficult to answer manually. In this case, we have only five trajectories, and each trajectory follows more or less the same path. If we have many more trajectories, and they follow clearly different paths like in Figure 1.1(b), this task becomes virtually impossible to do manually. Therefore, it is important that we can answer questions like this automatically. This is the main focus of this thesis; we consider various trajectory analysis tasks, and present efficient algorithms to perform them automatically. Note that we are mostly interested in the algorithms and techniques that can perform a certain task, rather than their results on a particular set of trajectories.

However, the large amount of data is not the only reason why answering the question from the previous paragraph is difficult. A more fundamental problem is that it is not clear what “similar” actually means. If we wish to compute the similarity between trajectories automatically, we need to define what it means for trajectories to be similar. This is true for all analysis tasks; before we can perform them automatically, we need to formalize the problem. An important aspect to this is how we actually define a trajectory itself. There are various models, which we review in Section 1.2. First, however, we provide a broad classification of trajectory analysis tasks. The tasks for which we provide efficient algorithms, are presented in more detail in Section 1.3.

1.1 Overview of Analysis Tasks

There are many different analysis tasks one might want to perform on trajectory data. Broadly, we distinguish between tasks that are of an exploratory nature, and tasks for which there is a more concrete and measurable goal. Tasks in the former category are usually solved using tools and techniques from visual analytics [11]. The tasks in the latter category may still be very specific to a particular application or data set. For example, “Find the longest time during which bird #15 was foraging.”. However, they can usually be reduced to one or more generic, application-independent tasks, as studied in the areas of geographic information science (GIS) and computational geometry. We restrict our attention to these generic, general-purpose tasks, and the algorithms and approaches to perform them.

Next, we give a short overview of several analysis tasks. Most tasks that we list involve multiple trajectories, although for some of them variations may also be applicable to a single trajectory. For example, we may be interested in finding and clustering movement patterns in a sub-trajectory [79].

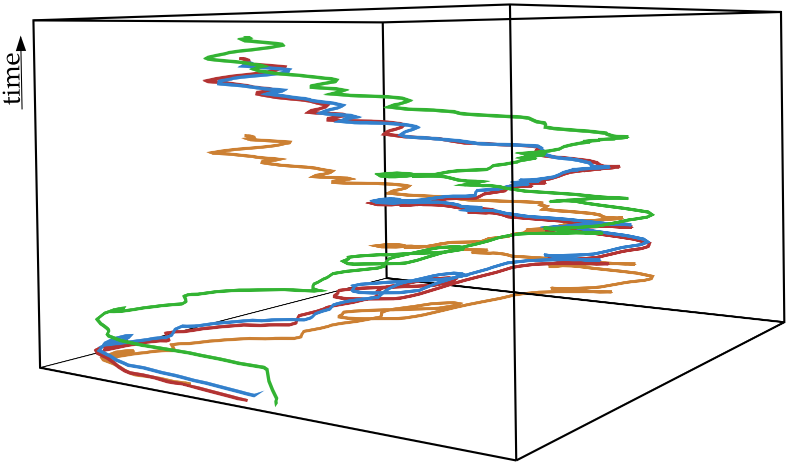


Figure 1.2: A view of four of the trajectories from Figure 1.1(a) that also shows time. Trajectories that appear similar (red and orange) in Figure 1.1(a) may be very different when considering time. An additional issue is if and how to align the trajectories (see e.g. the green trajectory).

Segmentation. Segmentation involves splitting or partitioning a trajectory into a number of sub-trajectories that have certain characteristics [4, 12, 31, 93, 121]. Important applications for segmentation are detecting mode of transportation, or classifying the (behavioral) state of the entity that produced the trajectory.

The characteristics used in segmentation can be based on trajectory attributes like speed, curvature, or acceleration, but often also include contextual data such as land-cover.

Similarity. The similarity of two trajectories can be captured by a function that takes two trajectories and returns a scalar: the higher the value, the higher the similarity. There are various aspects of the trajectories that need to be considered when defining a similarity measure. For example, should the temporal component of the trajectories be taken into account or not? If not, then we are basically considering shape similarity, a topic well-studied in pattern matching [5, 6]. If we do incorporate time, how should we handle trajectories of different duration? Figure 1.2 illustrates this issue.

Similarity is usually viewed as the reciprocal of distance. There are several common distance measures for trajectories. They usually differ in how they incorporate time. Examples include the Hausdorff distance [5], Fréchet distance [6], Dynamic Time Warping [83], time-focused distance [91], and edit distance [92]. What the right distance measure is depends on the application.

Besides similarity of whole trajectories, one could also define and compute similarity of sub-trajectories of two trajectories [27], or self-similarity of sub-trajectories within a single trajectory [79].

Clustering. Clustering is the process of partitioning trajectories into a (usually small) number of groups, or *clusters*, so that within each group the trajectories are similar, but across different groups they are dissimilar. Trajectory clustering has been well studied [57, 74, 91].

An important sub-task of clustering is *classification* [86]. Given a collection of clusters and a new trajectory, assign the trajectory to one of the clusters. The use of similarity measures to assist in classification and clustering is clear. Once a similarity measure is selected, many of the standard clustering methods for point data can be used for trajectory data as well, like single-linkage and complete-linkage clustering [75].

For clustering methods like k-means and k-medoids, one also needs a way to compute a representative for each cluster. That is, an algorithm that computes a specific "typical trajectory" that captures the common properties of the trajectories in the cluster.

Movement patterns. When we are analyzing a set of trajectories and are interested in interactions like joint movement or leadership, we speak of movement patterns in trajectories [41, 59, 76]. Several definitions of flocking [17, 62] and various other group movements [25, 85, 116] have been given and algorithms for these have been suggested.

Interesting places. From a collection of trajectories one can identify (small) places that are visited by many different trajectories, places that are used by a single moving entity for a long duration, or places where many trajectories change their movement behavior (e.g., they all pause). Such locations are called hotspots, popular places [16, 113], stationary regions [116], stay points, or stops [97, 102].

The analysis tasks that we study in more detail and for which we present algorithmic solutions are:

- Find a segmentation of a trajectory based on a non-monotone criterion.
- Find hotspots; regions in which the entity spent a large amount of time.
- Find all groups and the grouping structure. A group is a movement pattern in which sufficiently many entities move together during a sufficiently long time interval. In addition to the groups themselves we also find the relation between groups, e.g. a large group came into existence when two smaller groups merged.
- Find a central trajectory: a representative for a set of (similar) trajectories.

We discuss these tasks in more detail in Section 1.3.

1.2 A Trajectory

Before we give more detailed descriptions of the analysis tasks that we solve, it is important to define what a trajectory actually is. Throughout the existing literature, various models and definitions are used. A trajectory describes the movement of an entity, e.g. a person, car, bird, or hurricane, in time. Hence, in its purest form, a trajectory $\mathcal{T}$ is a continuous function mapping time to a subset of space. If we have a single trajectory, the time $\mathbb{T}$ can be represented by the interval $[0, 1] \subset \mathbb{R}$ without loss of generality. If we have multiple trajectories, they may have different global start and end times. This means that the trajectories become partial functions from time, usually some sub-interval of $[0, c]$, for some $c \in \mathbb{R}$, to space.

Which space we have to consider, and how we represent the location of an entity in that space, depends on the type of entities involved. For simplicity, we usually represent each entity by a single point. The space in which a person or hurricane moves is usually just the Euclidean plane $\mathbb{R}^2$. For birds and planes the height may also be important, hence, we may take space to be $\mathbb{R}^3$. Restricted or more complicated versions of space are also possible. For example, in applications involving car trajectories it is often assumed that cars can travel only on roads, so the space is the road network. In Chapter 6 we will see a more complicated space involving arbitrary (planar) obstacles.

We will restrict ourselves to trajectories that are piecewise linear. That is, throughout this thesis we will consider a trajectory to be a (continuous) piecewise linear function mapping time to space.

Even though the movement of the entities is often continuous, the corresponding trajectories are often collected and stored as a sequence of discrete time-stamped points. In this model one knows the position of the entity only at these times $t_1, .., t_\tau$. In some versions of this model, it is assumed that in between two consecutive time steps the entity moves along a straight line (similar to our model in which we assume that the trajectories are piecewise linear), in others the movement is really assumed to be discrete, and hence the entity jumps from its position at time t_i to its new position exactly at time t_{i+1} (this would correspond to the trajectory being a piecewise constant or partial function). Since the data is stored in a discrete format and it usually allows for simpler algorithms, this family of models is very popular in GIS and other applied fields [70, 77, 80, 88, 117].

A model that is becoming more popular recently is to explicitly model the uncertainty that is in trajectory data. At the discrete times $t_1, .., t_\tau$ at which we measured the exact position of the entity we represent the trajectory by a point. In between such times t_i and t_{i+1}, the entity may be anywhere within an

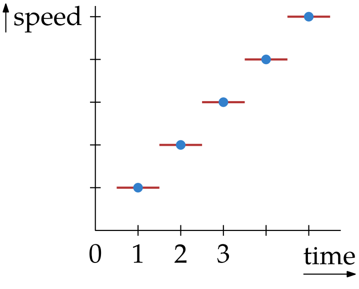


Figure 1.3: The discrete (blue points) and continuous (red line segments) representations of the speed s on the trajectory.

uncertainty region R_i [95]. To bound the size of the uncertainty regions this model then assumes that the moving entity has bounded speed. Extensions of this model also incorporate a probability distribution over these uncertainty regions. A well-known example of such a random movement model is the Brownian bridges model [69].

The choice of definition influences the results. It is important to note that the choice of definition can greatly influence the results of an analysis task. Consider the following example, in which we wish to compute a segmentation for the input trajectory (the task that we consider in more detail in Chapter 3). We construct two representations of the same trajectory –one in the continuous piecewise linear model, the other in the discrete model– for which the number of segments in an optimal segmentation differs significantly.

We compute a segmentation based on the standard deviation of the speed s on the trajectory. We construct the functions representing s directly, but note that it is easy to construct (the representations of) a trajectory realizing these speeds. For the discrete trajectory, we have the speeds only at discrete times $1, .., n$, hence s is defined only at those times. We set $s(t) = t$ for all $t \in 1, \ldots, n$. For the continuous model, we have the speed at every time $t \in \left[\frac{1}{2}, n + \frac{1}{2}\right)$. We set $s(t) = \left\lfloor t + \frac{1}{2} \right\rfloor$ for $t \in \left[\frac{1}{2}, n + \frac{1}{2}\right)$. Figure 1.3 illustrates these functions.

Suppose we segment on standard deviation with a threshold of 0.499, i.e. a sub-trajectory is a segment if the standard deviation of the speed on the sub-trajectory is at most 0.499. In the discrete model we must then take each index separately, because two consecutive indices give a standard deviation of 0.5. This only increases if we take longer segments. The segmentation on the continuous model can take as the first segment the first constant part with $s(t) = 1$ and nearly all of the second part with $s(t) = 2$. Similarly, the second segment can take the small remaining part where $s(t) = 2$, the entire part where $s(t) = 3$, and nearly all of the part where $s(t) = 4$ (but slightly less than before). In this example an optimal segmentation in the discrete model requires nearly twice as many segments as in the continuous model.

Terminology and notation. Throughout this thesis we use $\mathcal{T}_a$ to denote the trajectory of entity a. Recall that a trajectory is a function. Hence, $\mathcal{T}_a(t)$, for some time $t \in \mathbb{T}$, denotes the position of entity a at time t. We mostly consider point entities moving in $\mathbb{R}^2$, the position is then a point $\mathcal{T}_a(t) = p = (p_x, p_y)$ in the plane. We use $\|pq\|$ to denote the Euclidean distance between points p and q, and $\xi_{ab}(t) = \|\mathcal{T}_a(t)\mathcal{T}_b(t)\|$ for the (Euclidean) distance between entities a and b at time t. For ease of notation we will simply write $\mathcal{T}$ to mean the trajectory of entity a if there is only one entity, namely a. Similarly, we will sometimes simply write $a(t)$ to mean the position $\mathcal{T}_a(t)$ of entity a at time t.

We write $\mathcal{T}[s,t]$ to denote the sub-trajectory starting at time s and ending at time t; i.e. the function $\mathcal{T}$ restricted to the time interval $[s,t] \subseteq \mathbb{T}$. Our trajectories are piecewise linear, so they consist of maximal pieces $[s_i, t_i]$, with $t_i = s_{i+1}$, such that $\mathcal{T}[s_i, t_i]$ is a single linear function of the form $\gamma(t) = at + b$, with $a, b \in \mathbb{R}^2$. If a function consists of only one piece it is *simple*. We refer to the end-points of the pieces as *break points*.

When $\mathcal{T}$ has τ break points the image of $\mathcal{T}$ consists of $\tau - 1$ line segments in the plane, the *edges* of $\mathcal{T}$. The end-points of these edges (i.e. the positions corresponding to break points) are the *vertices* of $\mathcal{T}$.

Finally, if we have multiple entities (and thus trajectories) we use n to denote the number of entities and τ to denote the (maximum) number of vertices in a single trajectory. If we have only one trajectory we will use n to denote the number of vertices.

1.3 Outline

We present algorithms for four important trajectory analysis tasks: segmenting a trajectory, finding hotspots, finding groups, and finding a central representative trajectory. These tasks are presented in the main two parts, Parts II and III, of this thesis. Part I contains this introduction and the preliminaries, and Part IV contains some concluding remarks and an overview of future work.

Part II: Analysis Tasks for a Single Trajectory

Chapter 3: Non-Monotone Segmentation

In the trajectory segmentation problem we are given a trajectory that we have to subdivide into a minimum number of disjoint *segments* (sub-trajectories) that all satisfy a given criterion. A criterion is usually defined based on an attribute

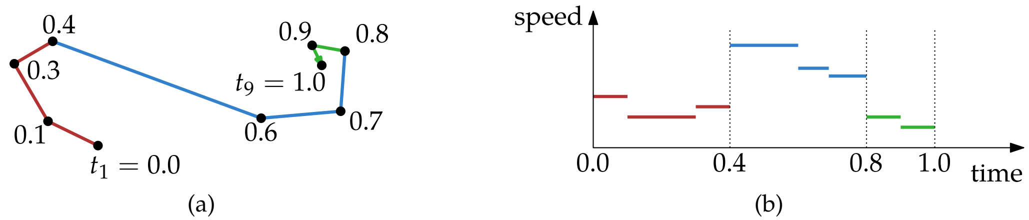


Figure 1.4: (a) A trajectory with vertices at times $t_1, .., t_9$. (b) The speed along the trajectory is a piecewise-constant function. We may divide the trajectory into three segments with "similar" speed: $[0.0, 0.4], [0.4, 0.8], [0.8, 1.0]$.

of the trajectory. For example, for the attribute "speed" we could segment the trajectory such that the minimum and maximum speed within any segment differ by at most h km/h, or by at most a factor of two (see Figure 1.4). This way, a segment represents a contiguous sub-trajectory for which a property is stable in some sense. Such sub-trajectories are often meaningful features of the trajectory. In the analysis of the trajectories of birds, for example, the goal is to extract the stretches where a certain activity is observed, such as soaring, directional flight, sleeping, etc. [99, 115]. Note that in general segments may start at any position along the trajectory, so a segment does not necessarily start at a trajectory vertex.

Previous research on similar segmentation problems has been done in animal movement studies [40, 99] and time-series analysis [9, 35, 112]. These solutions, however, provide no guarantees for individual segments in the segmentation. Instead, they are either heuristics or they optimize a global error criterion when a desired number of segments is specified. In the latter case, dynamic programming is a common approach [93]. The one exception is the research of Buchin et al. [31]. They show how to efficiently compute a segmentation for *monotone* criteria: criteria with the property that if they hold on a certain segment, they also hold on every sub-segment of that segment [31].

We present a broader study of the segmentation problem, and suggest a general framework for solving it, based on the *start-stop diagram*: a 2-dimensional diagram that represents all valid and invalid segments of a given trajectory. This yields two subproblems: (*i*) computing the start-stop diagram, and (*ii*) finding the optimal segmentation for a given diagram. We show that (*ii*) is NP-hard in general. However, we identify properties of the start-stop diagram that make the problem tractable, and give an efficient algorithm for this case.

We study two concrete non-monotone criteria that arise in practical applications in more detail. Both are based on a given univariate attribute function f over the domain of the trajectory. We say a segment satisfies an *outlier-tolerant* *criterion* if the value of f lies within a certain range for at least a given percentage of the length of the segment. For example, instead of requiring that the difference in speed within a segment is at most h, we require that this is the case for at least 95% of the time within each segment. A short burst in speed does not cause more segments in the segmentation with such a criterion. We say a segment satisfies a *standard-deviation criterion* if the standard deviation of f over the length of the segment lies below a given threshold.

We show that both the outlier-tolerant criterion and the standard-deviation criterion satisfy the properties that make the segmentation problem tractable. In particular, we compute an optimal segmentation of a trajectory based on the outlier-tolerant criterion in $O(n^2 \log n + kn^2)$ time, and on the standard-deviation criterion in $O(kn^2)$ time, where n is the number of vertices of the input trajectory and k is the number of segments in an optimal solution.

This chapter presents work that was published in:

[12] B. Aronov, A. Driemel, M. van Kreveld, M. Löffler, and F. Staals. "Segmentation of Trajectories on Non-Monotone Criteria." In: *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2013, pp. 1897–1911

Chapter 4: Finding Hotspots

We study one of the basic tasks in moving object analysis, namely the location of *hotspots*. A hotspot is a (small) region in which an entity spends a significant amount of time. We distinguish several versions of the problem of finding square-shaped hotspots. The problems we consider are:

1. The size of the square is fixed and we wish to find the placement that maximizes the time the entity spent inside. Here we allow the entity to leave the region and return to it later; all visits count for the duration.
2. We are given a duration and want to determine the smallest square and its placement so that the entity is inside for at least the given duration.
3. We consider problem 1, but now we are interested in *contiguous* presence inside the square, so only one of the visits to the square counts.
4. We are given a duration and want to determine the smallest square and its placement so that the entity is inside during a continuous time interval of the given duration.
5. We do not fix duration nor square size, but optimize a relative measure that is the ratio of the total duration and the square side length.

6. We consider optimizing a relative measure like in problem 5, but now for contiguous presence in the square.

Finding hotspots is useful in many applications, for example in finding the first-passage time of an animal, segmentation, clustering, and visualization.

In animal ecology, first-passage time is defined as the time taken for an animal to cross a circle of a given radius that is centered at some start time on the trajectory. It can be used as a measure for search effort along a trajectory, and a long first-passage time may indicate that there is a significant amount of food in an area [53, 78]. First-passage time is closely related to contiguous-time visit to a region, although the presence of food may be indicated better by total visit time of one or more animals.

When a trajectory is segmented into semantically meaningful parts, it may be important to use more advanced trajectory properties rather than just attributes like speed and heading. We may want to segment the trajectory based on the times at which it enters and leaves certain areas. These areas may be defined by the environment (e.g. based on land cover), but they can also depend on the trajectory itself. A straightforward example is an area in which the entity spends most of its time. In this situation we are interested in contiguous time within the area, so that crossing the area without halting does not lead to unnecessary segmentation.

In some clustering applications we may be interested in certain places where an entity spends some time, but not in the routes taken by the entity between these places. To compute the similarity of two trajectories in this situation we must identify these places, the hotspots, and compute the similarity based on the sequence of hotpots visited (for example using dynamic time warping [19]). A practical example of such a situation is in animal migration. We may find similarity in resting places more relevant than the routes traveled between these places.

Our methods can be used iteratively to enrich a trajectory data set with symbols that indicate that one or more entities spend much time there. For example, after locating a fixed-size square region where most time is spent, we can remove all pieces of trajectories inside that square, and iterate to find the next longest-visited square region. This is possible because our algorithms do not require trajectories to be contiguous or complete. We can then show the top-10 places in terms of duration of visits.

We do not assume that the hotspots are pre-defined; instead, we adopt the more generic view that no contextual data is present. This is in contrast to Alvares et al. [7] and Chawla and Verhein [116], who assume that a set of places or a superimposed grid is given that pre-defines potential regions of interest. Our work is different from research that uses the stop-and-move model [7, 97, 107], because we do not attempt to segment a trajectory at changes in movement type, nor do we try to semantically annotate a trajectory. Our methods allow the time of multiple visits to be added to obtain a hotspot: three visits of an hour can be considered more important than a single visit of two hours. Finally, our results are also different from research on popular places [16, 113] because the number of entities that visits the region plays no role. Benkert et al. [16] do not incorporate the duration of visits, so the retrieved popular places could be transit places used by many entities. Tiwari and Kaushik [113] consider only contiguous-time visits and choose their time intervals based on the trajectory vertices only, that is, they use the discrete trajectory model.

We solve the hotspot identification problem as an algorithmic optimization problem and provide running-time bounds. Our results are as follows. For a square hotspot, we can solve the contiguous-time versions in $O(n \log n)$ time, where n is the number of trajectory vertices. The algorithms for the total-time versions are roughly quadratic. Finding a hotspot in which the entity spends most time, relative to its size, takes $O(n^3)$ time. Extensions to different hotspot shapes are considered as well. We present our results for a single trajectory, but they immediately extend to the case of multiple trajectories.

This chapter presents work that was published in:

[63] J. Gudmundsson, M. van Kreveld, and F. Staals. "Algorithms for Hotspot Computation on Trajectory Data." In: *Proc. 21th International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '13. Orlando, Florida: ACM, 2013, pp. 134–143

Part III: Analysis Tasks for Multiple Trajectories

Chapter 5: The Trajectory Grouping Structure

We study algorithms for computing movement patterns. In particular, given a set of moving entities we wish to determine when and which subsets of entities travel together. When a sufficiently large set of entities travels together for a sufficiently long time, we call such a set a *group* (we give a more formal definition later). Groups may start, end, split and merge with other groups. Apart from the question of what the current groups are, we also want to know which splits and merges led to the current groups, when they happened, and which groups they involved. We wish to capture this group change information in a model that we call the *trajectory grouping structure*. We define such a structure using the *Reeb graph*, a concept from topology.

The informal definition above suggests that three parameters are needed to define groups: (i) a spatial parameter for the distance between entities; (ii) a temporal parameter for the duration of a group; and (iii) a count for the number of entities in a group. We will design our grouping structure definition to incorporate these parameters so that we can study grouping at different scales. We use the three parameters as follows: a small spatial parameter implies we are interested only in spatially close groups, a large temporal parameter implies we are interested only in long-lasting groups, and a large count implies we are interested only in large groups. By adjusting the parameters suitably, we can obtain more detailed or more generalized views of the trajectory grouping structure.

The use of scale parameters and the fact that the grouping structure changes at discrete events suggest the use of computational topology [47]. In particular, we use Reeb graphs to capture the grouping structure. A Reeb graph captures the structure of a two- or higher-dimensional scalar function, by considering the evolution of the connected components of the level sets. Reeb graphs have been used extensively in shape analysis and the visualization of scientific data [20, 48, 54]. The computation of Reeb graphs has received considerable attention in computational geometry and topology; an overview is given by Dey and Wang [39]. Recently, a deterministic $O(n \log n)$ time algorithm was presented for constructing the Reeb graph of a 2-skeleton of size n [103]. Edelsbrunner et al. [48] discuss time-varying Reeb graphs for continuous space-time data. Although we also analyze continuous space-time data, our Reeb graphs are not time-varying, but time is the parameter that defines the Reeb graph. Ge et al. [58] use the Reeb graph to compute a one-dimensional "skeleton" from unorganized data. In contrast to our setting, in their applications the data comes without a time component. They use a proximity graph on the input points to build a simplicial complex from which they compute the Reeb graph.

Our research is motivated by and related to previous research on flocks [17, 60, 61, 117], herds [70], convoys [77], moving clusters [80], mobile groups [72, 118] and swarms [88]. These concepts differ from each other in the way in which space and time are used to test if entities form a group: do the entities stay in a single disc or are they density-connected [51], should they stay together during consecutive time steps or not, can the group members change over time, etc. Only the herds concept by Huang, Chen, and Dong [70] includes the splitting and merging of groups. They consider the partition of the entities into clusters for two consecutive time stamps t_i and t_{i+1}. For a cluster A at time t_i and a cluster B at time t_{i+1} they classify the relation between A and B based on the number of members in $A \setminus B$, $A \cap B$, and $B \setminus A$. This classification includes join (merge) and leave (split).

We present the first complete study of trajectory group evolution, including combinatorial, algorithmic, and experimental results. Our research differs from and improves on previous research in the following ways. First, our model is simpler than herds and thus more intuitive. Second, we consider the grouping structure at continuous times instead of at discrete steps; all models but the flocks were studied only for discrete time steps. Third, we analyze the algorithmic and combinatorial aspects of groups and their changes. Fourth, we implemented our algorithms and provide evidence that our model captures the grouping structure well and can be computed efficiently. Fifth, we extend the model to incorporate robustness, that is, we show how brief interruptions of groups can be disregarded in the global structure, adding a notion of persistence to the structure.

Furthermore, we showcase the results of experiments in videos using data generated by the NetLogo flocking model [119, 120] and data from the Starkey project [100]. The Starkey data describe the movement of elk, deer, and cattle. Although there is no ground truth for the grouping structure in this data, the experiments show that the trajectory grouping structure is plausible and has the desired effects when changing the essential parameters.

This chapter presents work that was published in:

[28] K. Buchin, M. Buchin, M. van Kreveld, B. Speckmann, and F. Staals. “Trajectory Grouping Structure.” In: *Journal of Computational Geometry* 6.1 (2015), pp. 75–98

Chapter 6: Grouping under Geodesic Distance

We significantly extend the trajectory grouping structure from the previous chapter by incorporating contextual information. The entities generating the trajectories typically do not move in an infinitely large unrestricted space. Instead, they live in a world containing buildings, walls, lakes, etc., through which they cannot move. We incorporate obstacles into the trajectory grouping structure, and measure the distance between entities by their *geodesic distance*. The geodesic distance between two entities is the distance that needs to be traversed for one entity to reach the other entity. This approach gives a more natural notion of groups because it separates entities moving on opposite sides of obstacles like fences or water bodies. A threshold distance denoted by ε determines whether two entities are close enough to be in the same group. Hence we examine the number of times that a threshold distance occurs among n moving entities. Only threshold distances between the closest two entities of different groups matters, so we analyze the number of events of this type for various obstacle settings.

The combination of moving points and specific structures defined by these points has been a topic of major interest in computational geometry; for example, one of the main open problems in the area is the question “How many times can the Delaunay triangulation change its combinatorial structure in the worst case, when n points move along straight lines in the plane?” Other related research on movement in geometric algorithms concerns kinetic data structures. To our knowledge, this is the first work to combine continuously moving points with geodesic distances in the plane. We expect that our analysis will be of interest to other distance problems on moving points than the trajectory grouping structure, for example, in a similarity measure for two trajectories that incorporates obstacles.

We show that the trajectory grouping structure can be computed efficiently if obstacles are present and the distance between the entities is measured by geodesic distance. We bound the number of *critical events*: times at which the distance between two subsets of moving entities is exactly ε. In case the n entities move in a simple polygon along trajectories with τ vertices each we give an $O(\tau n^2)$ upper bound, which is tight in the worst case. In case of *well-spaced* obstacles we give an $O(\tau(n^2 + m\lambda_4(n)))$ upper bound, where m is the total complexity of the obstacles, and $\lambda_s(n)$ denotes the maximum length of a Davenport-Schinzel sequence of n symbols of order s. In case of general obstacles we give an $O(\tau(n^2 + m^2\lambda_4(n)))$ upper bound. We also present lower bounds that show that the last two upper bounds are close to optimal. Furthermore, for all cases we provide efficient algorithms to compute the critical events, which in turn leads to efficient algorithms to compute the trajectory grouping structure.

This chapter presents work that was published in:

[84] I. Kostitsyna, M. van Kreveld, M. Löffler, B. Speckmann, and F. Staals. “Trajectory Grouping Structure under Geodesic Distance.” In: *Proc. 31th annual Symposium on Computational Geometry*. SoCG ’15. Eindhoven, The Netherlands: Lipics, 2015

Chapter 7: Central Trajectories

The last task that we study is that of computing a representative trajectory for a set of trajectories. To this end we define a *central trajectory* $\mathfrak{C}$, which consists of pieces of the input trajectories, switches from one entity to another only if they are within a small distance of each other, and such that at any time t, the point $\mathfrak{C}(t)$ is as central as possible. We measure centrality in terms of the radius of the smallest disk centered at $\mathfrak{C}(t)$ enclosing all entities at time t, and discuss how the techniques can be adapted to other measures of centrality.

Finding a single representative trajectory for a set of trajectories is an important sub-task in clustering. Once a suitable clustering has been determined, the

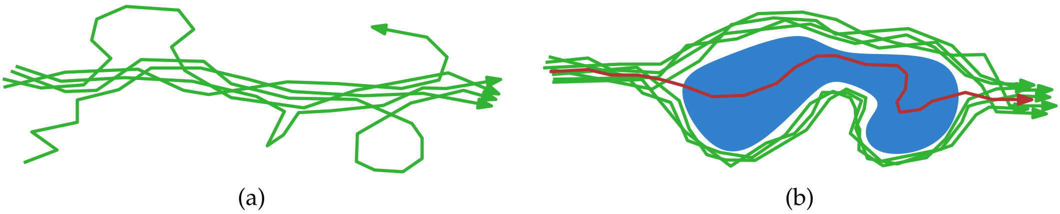


Figure 1.5: (a) Every trajectory has a peculiarity that is not representative for the set. (b) Taking, for example, the pointwise average of a set of trajectories may result in one that ignores context.

result needs to be stored or prepared for further processing. Storing the whole collection of trajectories in each cluster is often not feasible, because follow-up analysis tasks may be computation-intensive. Instead, we wish to represent each cluster by a signature: the number of trajectories in the cluster, together with a *representative* trajectory which should capture the defining features of all trajectories in the cluster.

Representative trajectories are also useful for visualization purposes. Displaying large amounts of trajectories often leads to visual clutter. Instead, if we show only a number of representative trajectories, this reduces the visual clutter, and allows for more effective data exploration. The original trajectories can still be shown if desired, using the details-on-demand principle in information visualization [105].

When choosing a representative trajectory for a group of similar trajectories, the first obvious choice would be to pick one of the trajectories in the group. However, it can be argued that no single element in a group may be a good representative, e.g. because each individual trajectory has some prominent feature that is not shared by the rest (see Figure 1.5(a)), or no trajectory is sufficiently in the middle all the time. On the other hand, it is desirable to output a trajectory that does consist of *pieces* of input trajectories, because otherwise the representative trajectory may display behavior that is not present in the input, e.g. because of contextual information that is not available to the algorithm. Figure 1.5(b) illustrates this issue: the representative trajectory based on the pointwise average goes through the lake, whereas all input trajectories go around it.

To determine what a good representative trajectory of a group of similar trajectories is, we identify two main categories: *time-dependent* and *time-independent* representatives. Depending on the application, we may be interested in a representative that includes the spatial as well as the temporal component of a trajectory –for example, when studying a flock of animals that moved together– or in a representative that includes only the spatial component –for example, when considering hikers that took the same route, but possibly at different times and speeds.

Recall that a trajectory is a function whose image is a (piecewise linear) curve. When we want a time-independent representative, we can select one based directly on the geometry or topology of this set of curves [26, 67]. When we want a time-dependent representative, we would like to have the property that at each time t our representative point $c(t)$ is a good representative of the set of points $P(t)$. To this end, we may choose any static representative point of a point set, and trace it over time. There are many choices for such a static representative point, for example, the Fermat-Weber point (which minimizes the sum of distances to the points in P), the center of mass (which minimizes the sum of squared distances), or the center of the smallest enclosing circle (which minimizes the distance to the farthest point in P).

We focus on time-dependent representatives. We measure centrality in terms of the radius of the smallest disk centered at $\mathfrak{C}(t)$ enclosing all entities at time t. Ideally, we would output a trajectory $\mathfrak{C}$ such that at any time t, $\mathfrak{C}(t)$ is the point (entity) that is closest to its farthest entity. Unfortunately, when the entities move in $\mathbb{R}^d$ for $d > 1$, this may cause discontinuities. Such discontinuities are unavoidable: if we insist that the output trajectory consists of pieces of input trajectories *and* is continuous, then in general, there will be no opportunities to switch from one trajectory to another, and we are effectively choosing one of the input trajectories again. At the same time, we do not want to output a trajectory with arbitrarily large discontinuities. An acceptable compromise is to allow discontinuities, or *jumps*, but only over small distances, controlled by a parameter ε. We note that this problem of discontinuities shows up for time-independent representatives for entities moving in $\mathbb{R}^d$, with $d \geq 3$, as well, because the traversed curves generally do not intersect.

Buchin et al. [26] consider the problem of computing a *median* trajectory for a set of trajectories without time information. Their method produces a trajectory that consists of pieces of the input. Agarwal et al. [1] consider trajectories with time information and compute a representative trajectory that follows the median (in $\mathbb{R}^1$) or a point of high *depth* (in $\mathbb{R}^2$) of the input entities. The resulting trajectory does not necessarily stay close to the input trajectories. They give exact and approximate algorithms. Durocher and Kirkpatrick [45] observe that a trajectory minimizing the sum of distances to the other entities is *unstable*, in the sense that arbitrarily small movement of the entities may cause an arbitrarily large movement in the location of the representative entity. They proceed to consider alternative measures of centrality, and define the *projection median*, which they prove is stable. Basu et al. [14] extend this concept to higher dimensions.

We first study the problem in $\mathbb{R}^1$, where we show that an optimal central trajectory $\mathfrak{C}$ representing n trajectories, each consisting of τ edges, has complexity $\Theta(\tau n^2)$ and can be computed in $O(\tau n^2 \log n)$ time. We then consider trajectories in $\mathbb{R}^d$ with $d \geq 2$, and show that the complexity of $\mathfrak{C}$ is at most $O(\tau n^{5/2})$ and can be computed in $O(\tau n^3)$ time. Additionally, we discuss how the techniques can be adapted to other measures of centrality.

This chapter is based on the work:

[114] M. van Kreveld, M. Löffler, and F. Staals. “Central Trajectories.” In: *CoRR* abs/1501.01822 (2015)

Chapter 2

Preliminaries

2.1 Parametric Search

We often deal with optimization problems of the following form. We are given a predicate $\mathcal{P}$ that is monotone in the real-valued parameter x, i.e. if $\mathcal{P}(x) = \textsc{True}$, then $\mathcal{P}(x') = \textsc{True}$ for all $x' \geq x$, and we wish to find the smallest value of x, say x^*, such that $\mathcal{P}(x^*) = \textsc{True}$. Megiddo's parametric search technique is a useful technique that can be used to solve such optimization problems [94]. As we use it on various occasions, we describe it here.

For the technique we need a decision algorithm $\mathcal{A}$ that can test predicate $\mathcal{P}$. We run $\mathcal{A}$ on the (unknown) optimum value x^*, while maintaining an interval I^* that is known to contain x^*. Initially $I^* = (-\infty, \infty)$. Every time the algorithm $\mathcal{A}$ encounters a comparison involving x^*, we obtain an equation $E(x^*)$ involving x^*.[1] We solve $E(x^*) = 0$, giving us a constant number of roots $x_1, .., x_c$. For each root, we run our decision algorithm $\mathcal{A}$ again. This tells us the values $\mathcal{P}(x_1), .., \mathcal{P}(x_c)$, which allows us to shrink the interval I^* with possible values for x^*, and get an answer for the comparison involving x^*. Thus, we can resume running $\mathcal{A}$ on x^*. Once this computation of $\mathcal{P}(x^*)$ finishes, that is, once we have encountered all comparisons made by algorithm $\mathcal{A}$, we can obtain x^* from I^* in constant time (either because I^* contains at most one value, the optimum, or by solving a simple equation).

If algorithm $\mathcal{A}$ takes $O(T)$ time, computing x^* takes $O(T^2)$ time in total: during the "main" evaluation of $\mathcal{P}(x^*)$, algorithm $\mathcal{A}$ encounters at most $O(T)$ comparisons, each one yielding a constant number of roots. Hence, in total we get $O(T)$ roots, on each of which we again run $\mathcal{A}$. Since this takes $O(T)$ time for each root, we spend $O(T^2)$ time in total:

▶ **Theorem 2.1** (Megiddo [94]). *Given a decision algorithm $\mathcal{A}$ that can compute $\mathcal{P}(x)$, for $x \in \mathbb{R}$, in $O(T)$ time, we can find the smallest value x^* such that $\mathcal{P}(x^*) = \textsc{True}$ in $O(T^2)$ time.*

---

[1] More formally, we need that $E(x^*)$ is a low-degree polynomial in x^*.

The running time required for this approach depends on how often we have to call the decision algorithm $\mathcal{A}$. So if running $\mathcal{A}$ is very expensive, then finding x^* this way is also very expensive. Megiddo therefore suggested the following.

We run $\mathcal{A}$ once for every root x_i of an equation $E(x^*)$. If we get many of such roots $x_1, .., x_m$ at once then we can handle this batch faster than the naive $O(mT)$ time, by using linear-time median finding and binary search. That is, we can process them in $O(m + T \log m)$ time. To get many roots at once, we need many independent equations, and thus independent comparisons. An easy way to get independent comparisons is to simulate running a parallel algorithm. That is, we replace the sequential algorithm $\mathcal{A}$ that we run on x^* by a parallel algorithm $\mathcal{A}_{\mathcal{P}}$. Suppose that $\mathcal{A}_{\mathcal{P}}$ runs in $O(T_P)$ time[2] using P processors, then simulating a single (parallel) step takes $O(P)$ time, and gives us $O(P)$ roots that we handle in $O(P + T \log P)$ time. Thus, if $\mathcal{A}_{\mathcal{P}}$ requires T_P steps, the entire procedure takes $O(T_P P + T_P T \log P)$ time in total.

▶ **Theorem 2.2** (Megiddo [94]). *Let $\mathcal{A}$ be a sequential decision algorithm that can compute $\mathcal{P}(x)$, for $x \in \mathbb{R}$, in $O(T)$ time, and let $\mathcal{A}_{\mathcal{P}}$ be a parallel decision algorithm for the same task that runs in $O(T_P)$ time using $O(P)$ processors. We can find the smallest value x^* such that $\mathcal{P}(x^*) = \textsc{True}$ in $O(T_P P + T_P T \log P)$ time.*

---

[2] Since we simulate running the parallel algorithm, it suffices to analyze the running time of $\mathcal{A}_{\mathcal{P}}$ in terms of the number of comparisons made. This also means that we can allow parallel algorithms that assume shared memory and allow simultaneous writes, etc.

Part II

Analysis Tasks for a Single Trajectory

Chapter 3

Non-Monotone Segmentation

In this chapter we present algorithms to segment a trajectory, that is, to partition the trajectory into a minimum number of *segments* such that each segment satisfies a given criterion [93]. A criterion is usually defined based on an *attribute* of the trajectory. An attribute is a function defined on the domain of the trajectory, for example speed, heading, or acceleration. So, a possible segmentation criterion could be: segment the trajectory such that the minimum and maximum speed within any segment differ by at most h km/h, or by at most a factor of two.

A criterion C is *monotone* if it has the property that if C is satisfied on some segment S of a trajectory, then it is also satisfied on any subsegment $S' \subseteq S$. Buchin et al. [31] provide a framework for segmenting trajectories based on a monotone criterion or a combination of several monotone criteria. Many criteria are monotone by nature. However, if the trajectory data is noisy, or a type of behavior is briefly interrupted, it is desirable to weaken the monotonicity requirement. For example, instead of requiring that the difference in speed within a segment is at most h, we could require that this is the case for at least 95% of the time within each segment. A short burst in speed does not cause more segments in the segmentation with such a criterion. Another example is using a threshold for the standard deviation of speed within a segment.

To segment a trajectory based on non-monotone criteria, we introduce the notion of the *start-stop diagram*. This allows us to split the problem into two subproblems: computing the start-stop diagram and computing an optimal segmentation for a given start-stop diagram. We show that the latter problem is NP-hard in general. However, a polynomial-time solution exists if the start-stop diagram has certain properties. We identify these properties in Section 3.2 and present an algorithm to compute an optimal segmentation. In Section 3.3, we consider how combining multiple criteria affects the properties of the start-stop diagram, and in Section 3.4 we present lower-bound constructions which show that several of our results from Section 3.2 are tight. We briefly discuss the segmentation problem when we consider trajectories in the discrete model (see the discussion from Section 1.2) in Section 3.5. We consider the issue of computing the start-stop diagram in Section 3.6; we show how to efficiently compute the start-stop diagram for two specific criteria. One such criterion, the *outlier-tolerant criterion*, requires that the minimum and maximum values of a piecewise-constant function f on each segment have at most a given difference, while allowing a certain percentage of outliers. The second criterion, the *standard-deviation criterion*, requires that on each segment the standard deviation of f is below a certain threshold on each segment. We show that these criteria satisfy the properties that make the segmentation problem tractable. In particular, a segmentation with the outlier-tolerant criterion can be computed in $O(n^2 \log n + kn^2)$ time, where n is the number of vertices in the input trajectory and k is the number of segments in an optimal segmentation. A segmentation with the standard-deviation criterion can be computed in $O(kn^2)$ time.

Although our framework was designed for trajectory segmentation, our techniques are also applicable to different problems. In particular, in Section 3.7 we discuss several variants of line simplification, and show that some of these fit our framework.

3.1 General Approach

We study the problem of segmenting a trajectory with respect to a criterion. Recall that a trajectory $\mathcal{T}$ is a continuous piecewise-linear function from time $\mathbb{T} = [0, 1]$ to $\mathbb{R}^2$ (or $\mathbb{R}^d$) and a sub-trajectory, also called *segment*, $\mathcal{T}[a, b]$ is the function restricted to the sub-interval $[a, b] \subseteq \mathbb{T}$. A criterion C is a function $C\colon \mathbb{T} \times \mathbb{T} \to \{\textsc{True}, \textsc{False}\}$, which is defined on all possible segments of $\mathcal{T}$. We say an interval $[a, b] \subseteq \mathbb{T}$ *satisfies* a criterion C if $C(a, b) = \textsc{True}$; in this case we call the segment *valid*. A partitioning of $\mathbb{T}$ (or of $\mathcal{T}$) into non-overlapping segments whose union covers $\mathbb{T}$ is called a *segmentation*. A segmentation of *size* k can be denoted by its segments $[\tau_0, \tau_1], [\tau_1, \tau_2], .., [\tau_{k-1}, \tau_k]$; $\tau_0 = 0$ and $\tau_k = 1$. A segmentation is *valid* if and only if all of its segments are valid, and *segmenting* a function refers to partitioning into valid segments. We say a valid segmentation is *minimal* (optimal) with respect to C if its size is minimum; the segmentation problem is to compute a valid minimal segmentation. We will often omit the word "valid" because only valid segmentations are useful.

A criterion is often based on an *attribute function* $f\colon \mathbb{T} \to \mathbb{R}$ that has the same domain $\mathbb{T}$ as $\mathcal{T}$, for example the function that maps time to the speed of the entity at that time. A segmentation of $\mathbb{T}$ based on the function f trivially induces a segmentation of $\mathcal{T}$, therefore we may use the term "segmenting f" to denote the resulting segmentation. We assume that f is piecewise constant and we call the points where the value of f changes the *breakpoints*. The portions between consecutive breakpoints, where f stays constant, are called *pieces*.

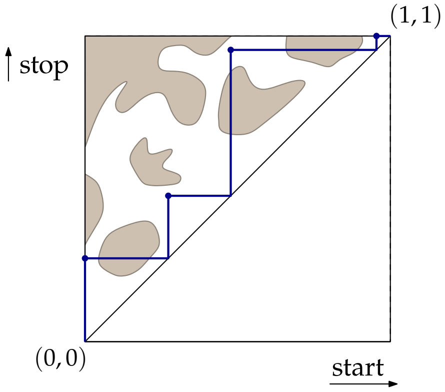


Figure 3.1: The start-stop diagram and a valid segmentation into four segments.

3.1.1 The start-stop diagram

To compute a minimal segmentation of f we define the *start-stop diagram.* Consider the parameter space of the set of sub-intervals of $\mathbb{T}$. For any candidate segment $[a, b]$, we associate the start parameter a with the horizontal axis and the stop parameter b with the vertical axis in the diagram. Any point (a, b) in this diagram, with $a < b$, represents a candidate segment. Thus, the set of candidate segments is represented by the points in the upper left triangle of the unit square. The set of points which represent valid segments defines the *free space* in the start-stop diagram. The remaining points constitute the *forbidden space*. A segmentation of $\mathbb{T}$ into a sequence of segments $[\tau_0, \tau_1], [\tau_1, \tau_2], [\tau_2, \tau_3], \ldots, [\tau_{k-1}, \tau_k]$; $\tau_0 = 0$ and $\tau_k = 1$, corresponds to a *staircase* in the start-stop diagram whose convex vertices correspond to the points (τ_i, τ_{i+1}) and whose concave vertices (τ_i, τ_i) lie on the main diagonal. A segmentation is valid if and only if all convex vertices lie in the free space. See Figure 3.1 for an example. Note that the start-stop diagram is reminiscent of the free space diagram used to compute the Fréchet distance [6]. However, in that problem one is interested in a path that stays inside the free space entirely. The size of a staircase, that is, the number of convex vertices, corresponds to the size of the segmentation. We will present efficient algorithms to segment a trajectory by computing the staircase in the start-stop diagram. However, we start with a negative result, showing that in general it is NP-hard to compute such a staircase efficiently.

The abstract segmentation problem. We define the *abstract segmentation problem* as follows. We are given a decomposition D of the triangle spanned by $(0, 0)$, $(0, 1)$ and $(1, 1)$ into points, a-monotone bounded-degree curve seg-

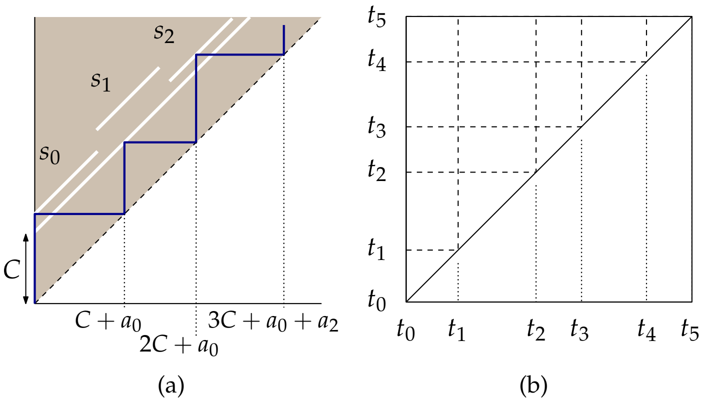


Figure 3.2: (a) Reduction from subset sum to the abstract segmentation problem. The staircase shows the sum a_0+a_2. (b) The subdivision of the start-stop diagram into cells according to the breakpoints of f.

ments (with horizontal axis a), and faces, each of which is marked either *free* or *forbidden*. Let n be the total complexity of D. In the resulting start-stop diagram, we need to find a staircase that represents a valid, minimal segmentation. We show that even testing the existence of a valid segmentation is NP-hard.

▶ **Theorem 3.1.** *The abstract segmentation problem is NP-hard.*

Proof. We reduce from Subset-Sum. Let $a_0, \dots, a_{n-1}$ be a set of positive integers and let B be the desired subset sum. We generate an instance of the abstract segmentation problem by constructing $n+1$ line segments in the start-stop diagram; these line segments are precisely the free space.

Let $A = \sum_{i=0}^{n-1} a_i$, $C = A + 1$ and $D = (n+1)C + B$. We generate a start-stop diagram of size $[0, D] \times [0, D]$, which we then scale to fit in the unit square. One line segment s of the free space has its endpoints at $(0, C)$ and at $(D-C, D)$ (see Figure 3.2(a)). For each a_i we create a line segment s_i with endpoints $(iC, (i+1)C + a_i)$ and $(iC + A, (i+1)C + a_i + A)$. This segment lies a_i units vertically above the first line segment s in the start-stop diagram. The placement of the segments is such that each staircase must choose to have its first convex vertex on s or s_0, its second convex vertex on s or s_1, and so on. Each subsequent step places the concave vertex C units further along the diagonal if s is chosen, and $C + a_i$ units if s_i is chosen. Each staircase that ends at (D, D) has exactly $n+1$ convex vertices, the last one necessarily on s. Furthermore, we can end at (D, D) if and only if the chosen s_i are such that the corresponding values a_i sum up to B.

The length of the segments s_i is chosen so that we can select s_i no matter what segments have been picked before. At the same time, the step size of at least C in every step makes sure that we cannot make more than one step on the same s_i. Clearly the reduction is polynomial. ☐

3.2 Solving the Segmentation Problem

Even though the general segmentation problem is NP-hard, we can identify properties of the start-stop diagram that make the problem tractable. We assume that we are given a start-stop diagram S (in the form specified in Section 3.1), and a decompositon of S into a grid of size $n \times n$. When f is a piecewise-constant function there is a natural decomposition of the start-stop diagram into a grid by the breakpoints of f, see Figure 3.2(b). In this case each cell of the grid corresponds to a set of candidate segments where the end points lie on fixed pieces of f. The cells incident to the diagonal are triangular and represent candidate segments with end points lying on the same piece.

Computing the reachable space. Recall that a minimal segmentation corresponds to a minimum-link staircase in the start-stop diagram. We define the *i-reachable space* R_i as the set of points on the diagonal that can be reached from $(0,0)$ by a valid staircase of size at most i. Whenever it causes no confusion, we identify the diagonal of the start-stop diagram with $\mathbb{T}$. In particular, we identify the (connected components of the) reachable space R_i with a set of *intervals* on $\mathbb{T}$. The number of intervals is the *complexity* of the reachable space. For a fixed i, the *incoming i-reachable intervals* of a cell $C_{i'j'}$ are the intervals of R_i intersected with the i'th column. We call the union of vertical (resp. horizontal) lines intersecting an interval X the vertical slab (resp. horizontal slab) induced by X. Consider the valid staircases of size at most $i+1$ which have their last convex vertex in such a vertical slab V_X. We call the connected components of the set of points on the diagonal that can be reached by such a staircase the *outgoing* $(i+1)$*-reachable intervals* produced by X.

The following algorithm describes an iterative procedure to compute R_i, for $i = 1, 2, ..$, starting with $R_0 = \{(0,0)\}$. The algorithm stops when it has found the smallest value of i for which R_i contains point $(1,1)$. Let this value be k. An actual minimum-link staircase, and hence a minimal segmentation, can then be extracted from the sets $R_0, .., R_k$.

```
Algorithm ReachableSpace($S$)
1.  $i \leftarrow 0; R_0 \leftarrow \{(0,0)\}$
2.  while $(1,1) \notin R_i$ do
3.      for each interval $X$ in $R_i$ do
4.          Consider the intersection of the free space in $S$ with the vertical
            slab induced by $X$ and project it horizontally back to the diagonal.
5.      The union of $R_i$ and these projections, over all $X$, forms $R_{i+1}$.
6.      $i \leftarrow i + 1$
7.  return $R_0, .., R_i$
```

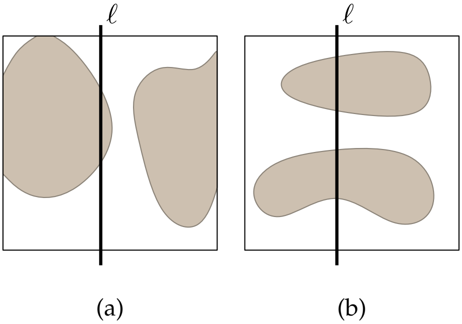


Figure 3.3: A verticonvex cell (a) and a tunnel cell (b).

We will make this algorithm more concrete when proving Theorem 3.3. The running time depends on the shape and the distribution of the forbidden space in each cell. Next, we distinguish different types of cells.

We call an object *verticonvex* if and only if its intersection with any vertical line ℓ is at most a single interval.[1] A cell (or row) in the start-stop diagram is *verticonvex* when the forbidden region within it is; see Figure 3.3(a). We call a cell a *tunnel cell* if any vertical line ℓ intersects the forbidden space in that cell in at most two intervals; see Figure 3.3(b) for an example. We will show that, for a given $n \times n$ start-stop diagram, we can compute a minimal segmentation in k segments in $O(kn^2)$ time if all cells are verticonvex, and in $O(k^2n^2)$ time if each row contains at most one tunnel cell and all other cells are verticonvex. If one allows cells where a vertical line can intersect the forbidden space in three or more intervals (i.e., "double-tunnels"), the problem becomes NP-hard as seen in the construction in the proof of Theorem 3.1. Even if we have only tunnel cells, but more than one tunnel cell in each row, the reachable space may have exponential complexity (as we will see in Proposition 3.11).

Note that a monotone criterion yields a monotone curve in the start-stop diagram. The region above the curve is the forbidden space, and the region below the curve is the free space. Hence, in such a start-stop diagram all cells are verticonvex.

3.2.1 Verticonvex cells only

Consider the algorithm ReachableSpace. The intersection of a vertical line with the free space in a verticonvex cell consists of at most two (possibly empty) intervals: one connected to the top, and one to the bottom of the row. The horizontal projection onto the diagonal preserves this property, and the union of the

---

[1] A verticonvex region is also called "x-monotone" in the literature (here it would be "a-monotone"); we choose to use the new term to avoid confusion with "monotone criteria."

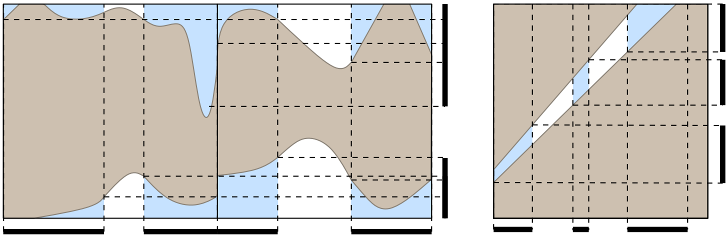

(a) (b)

Figure 3.4: (a) The $(i-1)$-reachable intervals for a row of verticonvex cells, and the at most two i-reachable intervals. In blue the possible positions for the i^{th} vertex in the staircase. (b) A tunnel cell may produce more i-reachable intervals.

projections of this type also consists of at most two intervals (see Figure 3.4(a)).

▶ **Observation 3.2.** *For any i and any row, the reachable space R_i produced by the incoming intervals in the verticonvex cells in the row consists of at most two intervals. One of these intervals is connected to the top of the row, and the other one is connected to the bottom of the row.*

We now prove:

▶ **Theorem 3.3.** *Given an $n \times n$ start-stop diagram in which the forbidden space in each cell is verticonvex and has constant complexity, we can compute a minimal segmentation in $O(kn^2)$ time, where k is the size of a minimal segmentation.*

Proof. Assume that for the given instance of the problem a valid segmentation exists. We specialize the procedure used in algorithm ReachableSpace. Recall that the reachable space can be encoded as a set of intervals, each of which corresponds to a connected component on the diagonal. Since all cells are verticonvex, Observation 3.2 implies that the i-reachable space, for any i, consists of $O(n)$ such intervals and as such we can store it as a set of $O(n)$ values. Given the reachable space R_i we compute the reachable space R_{i+1} in a row by row manner. By Observation 3.2, the $(i+1)$-reachable space restricted to any row consists of two intervals, one connected to the top and one to the bottom. We "grow" these two intervals, maintaining the top endpoint of the bottom one and the bottom endpoint of the top one while iterating over the cells in the current row and handling the i-reachable intervals incoming to every cell. Observation 3.2 implies that there are at most two incoming i-reachable intervals to any cell, since it is nothing more than the i-reachable space restricted to the column that contains the cell. Furthermore, the forbidden space in each cell has constant complexity. Therefore, we spend constant time per cell during this process and $O(n^2)$ time in total for computing R_{i+1} from R_i. We perform this step until the top-right corner $(1,1)$ is contained in R_{i+1}. Since this happens for $i+1=k$, this takes $O(kn^2)$ time. An optimal segmentation can be extracted by standard dynamic programming methods. ☐

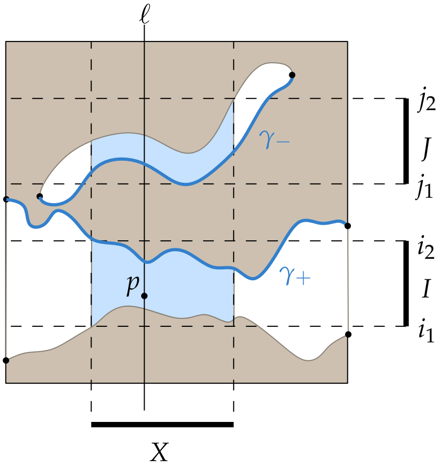


Figure 3.5: Illustration to the proof of Lemma 3.4. The vertices of the forbidden space are also shown.

3.2.2 At most one tunnel per row

Our algorithm ReachableSpace is efficient when all rows in the start-stop diagram are verticonvex. If a row is not verticonvex, it may produce more than two intervals; see Figure 3.4(b). Nonetheless, we show that if the start-stop diagram is not too complex—specifically, if each row contains at most one tunnel cell—the problem can still be solved efficiently. We start with the following technical result.

▶ **Lemma 3.4.** *Let C be a tunnel cell, and let $F \subseteq C$ be the forbidden space within C. An incoming interval X can produce at most one outgoing interval I such that (i) I is neither incident to the top nor the bottom of the row; and (ii) I does not intersect a horizontal line through a vertex of F.*

Proof. Assume, for the sake of contradiction, that X produces two intervals $I = [i_1, i_2]$ and $J = [j_1, j_2]$ with this property, with $i_2 < j_1$ (see Figure 3.5). Let H_I denote the horizontal slab induced by I, define H_J similarly, and let V_X denote the vertical slab induced by X. Since I does not intersect a horizontal line through a vertex of F, H_I does not contain any vertices. The same holds for H_J.

Since X produced the outgoing interval I, there must be at least one point p of free space in $V_X \cap H_I$. Let ℓ be the vertical line through p, and let p_+ be the first intersection of ℓ with the boundary of free space above p; p_+ may coincide with p. Notice that p_+ must lie in the interior of a boundary edge γ_+ of free space: p_+ may not be a vertex, as there are no vertices in H_I. The edge γ_+ may not cross the top boundary of $V_X \cap H_I$, for otherwise I would be larger.

Repeating the construction for segment J, we obtain a free-space point $q' \in V_X \cap H_J$ on a different vertical line ℓ'. Arguing as above, but now working downwards, we obtain a boundary edge γ_- of free space that does not cross the bottom edge of $V_X \cap H_J$. Since γ_- extends all the way across $V_X \cap H_J$, it must cross ℓ within that rectangle and lie below some free space point $q \in \ell \cap H_J$.

We have obtained a contradiction, as now ℓ intersects F at least three times: once below H_I (since I is not incident to the bottom of the row), once between γ_+ and γ_- and once above H_J (since J is not incident to the top of the row), so C is not a tunnel cell. $\square$

▶ **Lemma 3.5.** *Let C be a tunnel cell, and let $F \subseteq C$ be the forbidden space within C. Suppose C has m incoming i-reachable intervals and u is the complexity of the union of the outgoing $(i+1)$-reachable intervals produced by them. Then $u \leq c + m + 2$, where c is the number of vertices of F.*

Proof. We draw a horizontal line h_v through every vertex v of F. Let H denote this set of lines. We charge every interval in the union of the outgoing intervals that intersects such a line h_v to v. This way, each vertex gets charged at most once, since two intervals that would charge the same line cannot be disjoint. Thus, we have at most c charges of this type.

By Lemma 3.4, each incoming interval can produce at most one outgoing interval that does not intersect a line of H and is incident neither to the top nor to the bottom of the row. Therefore, we can charge those to the incoming intervals with at most m charges.

The remaining intervals in the union are incident either to the top or the bottom of the row, and of those there can be at most two in total. $\square$

▶ **Lemma 3.6.** *In an $n \times n$ start-stop diagram in which (i) the forbidden space in each cell has constant complexity, (ii) each row contains at most one tunnel cell, and (iii) the remaining cells are verticonvex, R_i consists of at most $O(in)$ intervals.*

Proof. Consider the i-reachable space R_i restricted to one row of the grid. It consists of the endpoints of all valid staircases that have their last convex vertex in a cell of this row.

We distinguish two types of staircases: (a) those that have their last convex vertex in a verticonvex cell and (b) those that have it in a tunnel cell. By Observation 3.2, the i-reachable space that is formed by the staircases of type (a) has constant complexity.

As for the staircases of type (b), they all have their last convex vertex in the same cell since there is only one tunnel cell per row. By Lemma 3.5, and since the forbidden space in every cell has constant complexity, the i-reachable space formed by those staircases consists of $m + O(1)$ intervals, where m is the complexity of the $(i-1)$-reachable space R_{i-1} restricted to the column that the tunnel cell lies in.

Thus, we have the following recurrence for the complexity of the i-reachable space restricted to one row: $C(i) \leq C(i-1) + O(1)$. Clearly, $C(1) \leq 3$, and therefore, we have that $C(i) = O(i)$. There are n rows, thus the total complexity of the i-reachable space is $O(in)$. ☐

Using essentially the same algorithm as before we now obtain:

▶ **Theorem 3.7.** *Given an $n \times n$ start-stop diagram in which (i) the forbidden space in each cell has constant complexity, (ii) each row contains at most one tunnel cell, and (iii) the remaining cells are all verticonvex, we can compute a minimal segmentation in $O(k^2n^2)$ time, where k is the size of a minimal segmentation.*

Proof. We again use the algorithm ReachableSpace and traverse the start-stop diagram as described in the proof of Theorem 3.3. Since the complexity of the forbidden space is constant in each cell, we can list a set of m incoming intervals and produce the at most $m + O(1)$ outgoing intervals in $O(m)$ time. It follows that we can compute R_i from R_{i-1} in $O(in^2)$ time, and thus the total running time is $O(k^2n^2)$. ☐

3.3 Combining Criteria

We can also consider segmenting a trajectory based on a combination of multiple criteria. For example, we want segments such that two criteria hold simultaneously, or where at least one criterion holds. We can combine criteria into new ones by taking conjunctions, disjunctions, and negations, and in general we can build any boolean combination this way.

Again, we assume that there exists a grid decompositon of size $n \times n$ or smaller of the start-stop diagram for each of the criteria. To obtain the start-stop diagram for the criterion $C_1 \wedge C_2$, we simply overlay their grids and take the union of the forbidden space F_1 of C_1 and the forbidden space F_2 of C_2. Similarly, the forbidden space for $C_1 \vee C_2$ is the intersection of F_1 and F_2.

Buchin *et al.* [31] observe that the conjunction or disjunction of monotone criteria is again monotone. Similarly, we observe that the start-stop diagram of a disjunction of two *verticonvex criteria*, i.e., criteria for which the start-stop diagram contains only verticonvex cells, again contains only verticonvex cells. Hence the disjunction of two verticonvex criteria is again a verticonvex criterion. Since a monotone criterion is also verticonvex, the disjunction of a monotone criterion with a verticonvex criterion results in a verticonvex criterion as well.

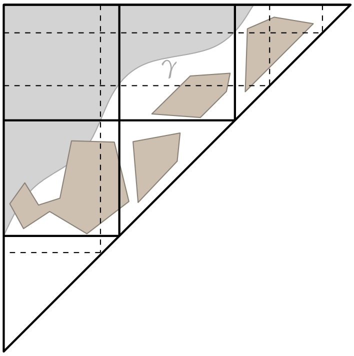


Figure 3.6: The start-stop diagram for the conjunction of a monotone and a verticonvex criterion. The forbidden space resulting from the monotone criterion is shown in gray. We further subdivide the start-stop diagram (dashed lines) such that the boundary of this region γ intersects at most one cell in row and in any column.

For the conjunction of two verticonvex criteria we take the union of their forbidden spaces. Unfortunately, this can lead to non-verticonvex cells. However, we can show that if one of the criteria is monotone, we can slightly modify the grid of the combined start-stop diagram such that it contains at most one tunnel cell in each row and the remaining cells are verticonvex. By Theorem 3.7 we can therefore still compute a minimal segmentation efficiently.

▶ **Proposition 3.8.** *Let C_1 be a monotone criterion, let C_2 be a verticonvex criterion, and let the overlay of their grids have size $n \times n$. If the complexity of the forbidden space in every cell is constant, then we can compute a minimal segmentation with respect to the criterion $C_1 \wedge C_2$ in $O(k^2 n^2)$ time, where k is the size of a minimal segmentation.*

Proof. We argue that there exists an $O(n) \times O(n)$ refinement of the overlay of the two grids such that (*i*) every row contains at most one tunnel cell, and (*ii*) the remaining cells are verticonvex. The claim then follows from Theorem 3.7.

Recall that a monotone criterion corresponds to an *ab*-monotone curve γ in the start-stop diagram (with axes a and b), in which all points above the curve form the forbidden space, and all points below the curve lie in the free space. We subdivide the grid such that γ intersects at most one cell in any row, and at most one cell in any column (see Figure 3.6). Since γ is *ab*-monotone it intersects an original grid line at most once; this means we add at most $O(n)$ grid lines. The complexity of the forbidden space in each cell is constant.

Clearly, the cells of the refined grid that are not intersected by γ are verticonvex: everything above γ is forbidden space, and the forbidden space in other cells originates only from a verticonvex criterion. The cells that are intersected by γ are (at worst) tunnel cells since every vertical line intersects the forbidden space at most twice: at most once for the forbidden space resulting from the verticonvex criterion, and at most once for the forbidden space above γ. Thus, we have at most one tunnel cell per row. □

Note that essentially the same approach can also be used to solve the problem for the conjunction of a verticonvex criterion and the negation of a monotone criterion:

▶ **Corollary 3.9.** *Let* C_1 *be a monotone criterion, let* C_2 *be a verticonvex criterion, and let the overlay of their grids have size* $n \times n$*. If the complexity of the forbidden space in every cell is constant, then we can compute a minimal segmentation with respect to the criterion* $\neg C_1 \wedge C_2$ *in* $O(k^2 n^2)$ *time, where* k *is the size of a minimal segmentation.*

3.4 Lower bounds

In this section we present lower bounds on the complexity of the reachable space for several types of start-stop diagrams. In particular, we show that the k-reachable space in a start-stop diagram representing the conjunction of a monotone and a verticonvex criterion may have complexity $\Omega(kn)$, and that if we have more than one tunnel cell per row then the k-reachable space may have exponential complexity.

Gadgets. We construct the start-stop diagrams by gluing together individual cells. Given a start-stop diagram, let C_{ij} be the cell in column i and row j. We identify C_{ij} with the unit square $[0,1] \times [0,1]$, which allows us to describe C_{ij} using a combination of gadget cells, or *gadgets* for short. A gadget acts as a function, mapping reachable points in the incoming intervals of the cell to the reachable points in the outgoing intervals. It will, however, be more convenient to model each gadget α as a function f_α mapping incoming points $x \in [0,1]$ to *a set* of points $f_\alpha(x) \subseteq [0,1]$. This set $f_\alpha(x)$ corresponds to the free space within the cell on the vertical line at x.

We write $[x]$ for the interval (set) consisting of just x and $f(X)$ for the image of a subset $X \subseteq [0,1]$ under f. The gadgets that we use are then $f_\emptyset(x) = \emptyset$, to model cells that are completely forbidden, and the gadgets shown in Figure 3.7. Note that gadgets (a), (b) and (c) are tunnel cells, while gadgets (d) and (e) are verticonvex cells.

When combining cells C_{ij} and $C_{(i+1)j}$ into the final start-stop diagram, we use the description of $C_{(i+1)j}$ for their shared border. Similarly, we give preference to $C_{i(j+1)}$ over C_{ij} for their shared border. Furthermore, let f_{ij} be the function describing C_{ij}. For row j, we then define

$$R_k^j = \bigcup_{i=1}^{j-1} f_{ij}\left(R_{k-1}^i\right) \text{ and } R_1^j = f_{0j}(0). \tag{3.1}$$

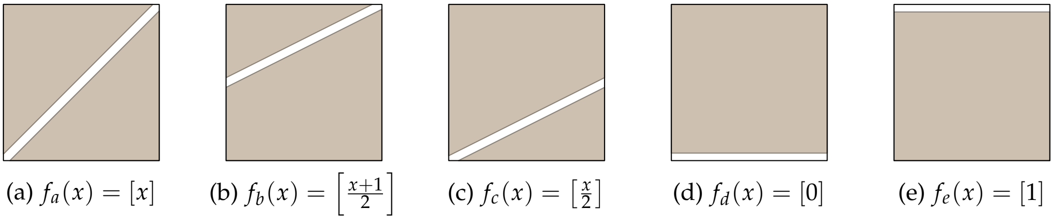

(a) $f_a(x) = [x]$ (b) $f_b(x) = \left[\frac{x+1}{2}\right]$ (c) $f_c(x) = \left[\frac{x}{2}\right]$ (d) $f_d(x) = [0]$ (e) $f_e(x) = [1]$

Figure 3.7: The gadget cells used in the lower bound constructions.

The set R_k^j denotes the k-reachable space restricted to the jth row (expressed in the coordinates of the row). We can obtain the reachable space R_k, by reparameterizing each R_k^j and taking the union over all j.

In all our constructions the triangular cells incident to the diagonal are completely forbidden. So a staircase ending in the jth row can have at most $j-1$ steps. Thus, $R_k^j = R_{k-1}^j$ for $k \geq j$. (Recall that the k-reachable space corresponds to the valid staircases with at most k steps.)

At most one tunnel per row. The running time in Proposition 3.8 (and Theorem 3.7) is a factor k worse than that in Theorem 3.3. We now give an example of an $n \times n$ start-stop diagram resulting from the conjunction of a monotone and a verticonvex criterion in which the complexity of the k-reachable space is $\Omega(kn)$. In this start-stop diagram, every cell has constant complexity, every row contains at most one tunnel cell and all the remaining cells are verticonvex. Thus, the lower bound of Proposition 3.10 below matches the worst-case upper bound we give in Lemma 3.6. This explains the extra factor k in the running time of our algorithm.

▶ **Proposition 3.10.** *There exists an $n \times n$ start-stop diagram such that (i) the forbidden space is the union of the region above a monotone curve and one verticonvex region per cell, (ii) the forbidden space in each cell is polygonal and has constant complexity, and (iii) the complexity of the k-reachable space is $\Omega(n^2)$, where k is the size of an optimal segmentation.*

Proof. We construct a start-stop diagram cell by cell using the gadgets described above. On the basis of the description of the reachable space in Equation (3.1), we construct the start-stop diagram such that the reachable space satisfies the following recurrence:

$$R_k^j = f_d\left(R_{k-1}^{j-1}\right) \cup f_b\left(R_{k-1}^{j-1}\right) \text{ and } R_1^2 = [0] \cup \left[\frac{1}{2}\right]. \tag{3.2}$$

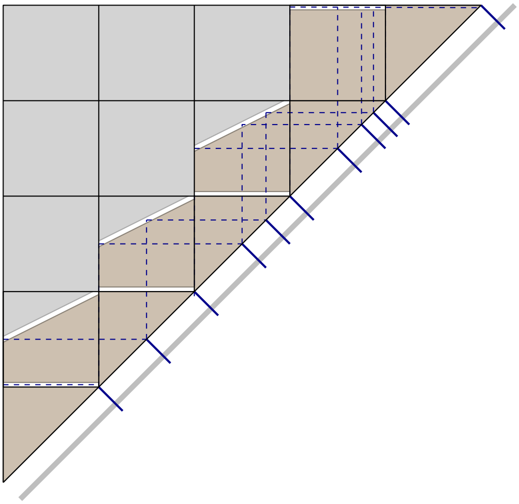


Figure 3.8: A start-stop diagram of the conjunction of a monotone criterion and a verticonvex one that results in a reachable space of complexity $\Omega(n^2)$.

To accomplish this, we place the gadgets as follows. For cells C_{ij}, with $j = i+1$ and $j < n$, we use an overlay of gadgets (b) and (d) from Figure 3.7. For $j = n$ and $i = n-1$, we use gadget (e). All other cells are forbidden. The overall diagram is shown in Figure 3.8. It can be seen as a conjunction of a monotone criterion and a verticonvex one.

It is easy to prove by induction that the following holds for $j \in 2, \dots, n-1$:

$$R^j_{j-1} = [0] \cup \bigcup_{i=1}^{j-1} \left[\frac{2^i - 1}{2^i} \right]. \tag{3.3}$$

In our construction, the minimum size of an optimal segmentation is at least $n-2$. This follows from the fact that the cells C_{ij} with $j \geq i+2$ are completely forbidden. Furthermore, by Equation (3.3), for any j, all intervals in R^j_{j-1} are pairwise disjoint. Note that also across the rows, the intervals are disjoint. Therefore, the total number of these intervals over all rows is a lower bound on the complexity of the k-reachable space, where k is the size of an optimal segmentation. Thus,

$$\sum_{j=1}^{n-1} \left| R^j_k \right| \geq \sum_{j=2}^{n-1} \left| R^j_{j-1} \right| = \sum_{j=2}^{n-1} j = \Omega(n^2),$$

where $|\cdot|$ denotes the description complexity[2] of the set. This completes the proof. □

---

[2] Our sets are finite unions of intervals. By the *description complexity* of a set we mean the minimum number of intervals required to write the set as a union.

More than one tunnel per row. Next, we show that if we allow two tunnel cells per row, then the reachable space can have exponential complexity. The idea of the construction is the following. We use a block of three gadgets (a), (b) and (c) from Figure 3.7. We arrange the three gadgets so that the complexity of the reachable space is doubled every two rows. The construction is illustrated in Figure 3.9. Indeed, the tunnel in gadget (a) exactly copies the reachable space in column i to column $i+1$, and the tunnels in gadgets (b) and (c) "compress" both copies of the reachable intervals and project them onto column $i+2$. Since the reachable space in the first column consists of exactly one point this leads to a reachable space in the last column that consists of an exponential number of isolated points.[3]

▶ **Proposition 3.11.** *There exists an $n \times n$ start-stop diagram such that (i) the forbidden space in each cell has constant complexity, (ii) each row contains at most two tunnel cells, (iii) the remaining cells are all verticonvex, and (iv) the complexity of the k-reachable space R_k is $\Omega(2^{n/4})$, where k is the size of an optimal segmentation.*

Proof. We again use the gadgets from Figure 3.7. For cell C_{12} we use gadget (d), for $C_{(n-1)n}$ we use gadget (e). For the cells $C_{i(i+1)}$ we use gadget (a) when i is even, and gadget (c) when i is odd. For $C_{i(i+2)}$ we use gadget (b). All remaining cells are forbidden. The overall construction is shown in Figure 3.9. We express the reachable space in the form of Equation (3.1).

$$\begin{aligned} R^j_k &= f_c\left(R^{j-1}_{k-1}\right) \cup f_b\left(R^{j-2}_{k-1}\right) \\ &= f_c\left(f_a\left(R^{j-2}_{k-2}\right)\right) \cup f_b\left(R^{j-2}_{k-1}\right) \qquad \text{and } R^2_2 = R^2_1 = [0]. \end{aligned} \tag{3.4}$$

We claim that the following holds for even j larger than 2:

$$R^j_{j-1} = \bigcup_{i=0}^{2^{j/2-1}-1} \left[\frac{i}{2^{j/2-1}}\right]. \tag{3.5}$$

We prove this by induction. For $j=2$ this is clearly true. For $j \to j+2$, using Equation (3.4) we obtain

$$R^{j+2}_{j+1} = f_c\left(R^j_{j-1}\right) \cup f_b\left(R^j_j\right).$$

---

[3] It may appear that in this construction one could simply refine the grid by adding a horizontal grid line through the center of every row and obtain a start-stop diagram where every row has at most one tunnel. This would seem to contradict Lemma 3.6, which states that in such a start-stop diagram the i-reachable space has complexity $O(in)$. However, since the grid has to be symmetric, we also have to add a vertical line through the point where the new horizontal line crosses the main diagonal. It so happens that the resulting vertical line again cuts a tunnel cell into two, resulting in two tunnel cells per row. Incrementally adding lines in this manner until there is at most one tunnel in every row leads to a grid of exponential size.

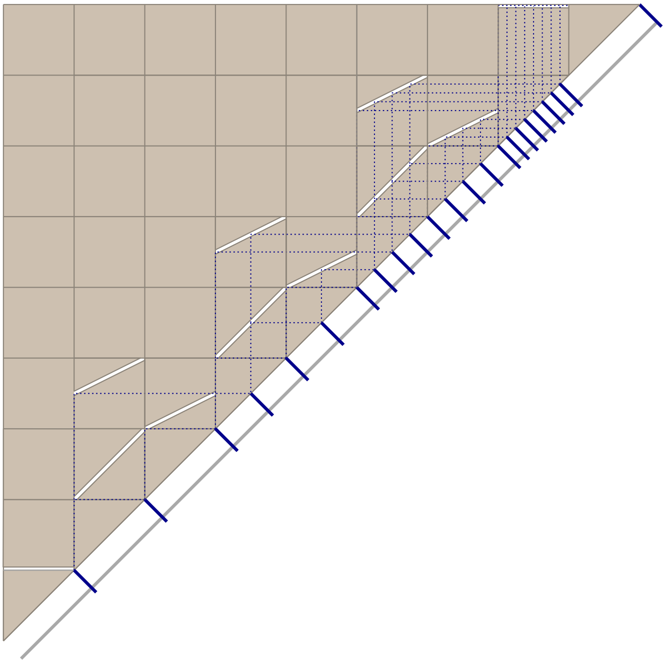


Figure 3.9: A start-stop diagram with at most two tunnels per row that results in a reachable space of exponential complexity.

By the induction hypothesis for R^{j}_{j-1} and since $R^{j}_{j} = R^{j}_{j-1}$:

$$\begin{aligned}
R^{j+2}_{j+1} &= \bigcup_{i=0}^{2^{j/2-1}-1} \left[\frac{i}{2^{j/2}}\right] \cup \bigcup_{i=0}^{2^{j/2-1}-1} \left[\frac{i}{2^{j/2}} + \frac{1}{2}\right] \\
&= \bigcup_{i=0}^{2^{j/2-1}-1} \left[\frac{i}{2^{j/2}}\right] \cup \bigcup_{l=2^{j/2-1}}^{2^{j/2}-1} \left[\frac{l - 2^{j/2-1} + 2^{j/2-1}}{2^{j/2}}\right] \\
&= \bigcup_{i=0}^{2^{j/2}-1} \left[\frac{i}{2^{j/2}}\right].
\end{aligned}$$

This proves Equation (3.5) for even j. Since the intervals generated by this sequence are pairwise disjoint, the complexity of R^{j}_{j-1} is at least $2^{j/2}$. A valid segmentation has to have at least $\lfloor n/2 \rfloor$ steps, since all cells C_{ij} with $j \geq i+3$ are completely forbidden. Thus, the complexity of the k-reachable space, where k is the size of the optimal segmentation, is lower bounded by the complexity of $R^{\lfloor n/2 \rfloor}_{\lfloor n/2 \rfloor - 1}$. Therefore, the complexity is at least $\Omega(2^{n/4})$. □

Incremental Reachability. One might argue that to solve the segmentation problem, we only need to maintain the incremental reachable space, i.e., the

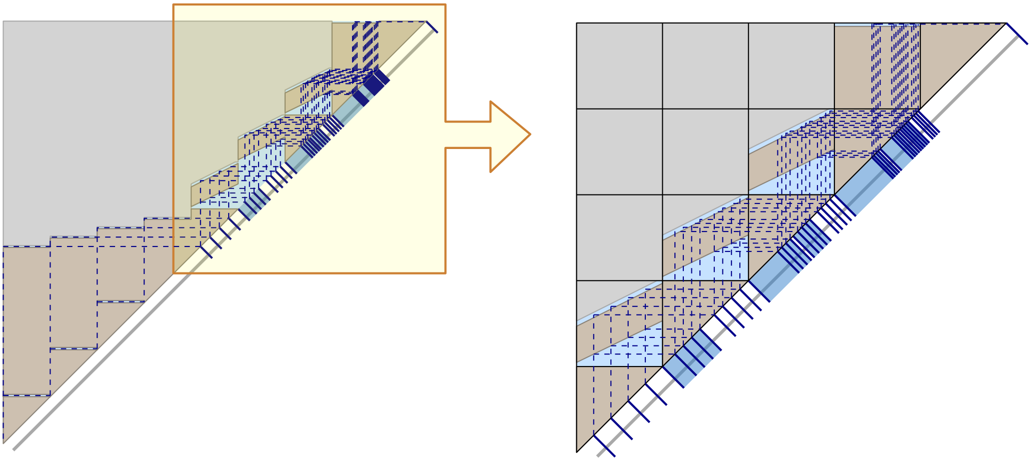


Figure 3.10: A start-stop diagram of the conjunction of a monotone criterion and a verticonvex one that results in a high complexity incremental reachable space.

subset of the diagonal $R_i \backslash R_{i-1}$ reachable with exactly i steps. Next we show that this set can also have high complexity. To this end we extend the construction given in Proposition 3.10.

▶ **Observation 3.12.** *There exists an $n \times n$ start-stop diagram such that (i) the forbidden space is the union of the region above a monotone curve and one verticonvex region per cell, (ii) the forbidden space in each cell is polygonal and has constant complexity, and (iii) the complexity of $R_i \setminus R_{i-1}$ is $\Omega(i^2)$ for $i \in 2, \ldots, \lfloor n/2 \rfloor$.*

Proof. In this construction we use the gadgets from Figure 3.7, as well as an additional (type of) gadget similar to gadget (d). This new gadget $g_i = [x_i] = [2i/n]$, for $i \in 1, \ldots, \lfloor n/2 \rfloor$, is essentially a constant function with value x_i. Assume for simplicity of presentation that n is odd and let $m = \lceil n/2 \rceil$ be the index of the middle row. We place the cell with gadget function g_i in the ith column and mth row. Furthermore, there are gadgets of type (d) in the ith column of row $i + 1$ for $i \in 0, \ldots, m - 1$. The diagram for columns $i \geq m$ is a copy of the construction used in Proposition 3.10, which we modify as follows (see Figure 3.10). Instead of using an overlay of gadget (b) and (d), we overlay gadget (b) with a modification of gadget (c), which maps x to an interval instead of a single value. The gadget function is given by

$$f_g(x) = \left[0, \frac{x}{2}\right].$$

For the reachable space in the m^{th} row, we have

$$R_i^m = \bigcup_{j=1}^{i} \left[x_j\right].$$

For the upper part of the diagram (i.e., columns with index greater than m) we have a recurrence similar to that of Equation (3.2):

$$R_k^{m+q} = f_g\left(R_{k-1}^{m+q-1}\right) \cup f_b\left(R_{k-1}^{m+q-1}\right).$$

It is easy to prove by induction on q that the following holds:

$$R_{i+q}^{m+q} = \bigcup_{l=0}^{q-1}\left[1-\frac{1}{2^l}\,,\,1-\frac{1}{2^{l+1}}-\frac{1-x_i}{2^q}\right] \cup \bigcup_{j=1}^{i}\left[1-\frac{1-x_j}{2^q}\right],$$

for $i \in 1, \dots, \left\lfloor \frac{n}{2} \right\rfloor$ and $m+q<n$.

Now we are interested in the incremental reachable space, which can be written as follows:

$$R_{i+q}^{m+q} \setminus R_{(i-1)+q}^{m+q} = \bigcup_{l=0}^{q-1}\left[1-\frac{1}{2^{l+1}}-\frac{1-x_{i-1}}{2^q}\,,\,1-\frac{1}{2^{l+1}}-\frac{1-x_i}{2^q}\right] \cup \left[1-\frac{1-x_i}{2^q}\right].$$

Since the intervals in the above equation are pairwise disjoint (since $x_i \in (0,1)$ for all i), we have that

$$\left|R_{i+q}^{m+q} \setminus R_{(i-1)+q}^{m+q}\right| = q+1,$$

where $|\cdot|$ denotes the description complexity of the set. Therefore we can write

$$\sum_{p=1}^{m+k-1}\left|R_k^p \setminus R_{k-1}^p\right| \geq \sum_{p=m}^{m+k-1}\left|R_k^p \setminus R_{k-1}^p\right| = \sum_{q=0}^{k-1}\left|R_k^{m+q} \setminus R_{k-1}^{m+q}\right| = \sum_{q=0}^{k-1}(q+1) = O(k^2),$$

using $k = i+q$ for $i \in 1, \dots, \left\lfloor \frac{n}{2} \right\rfloor$. This completes the proof. □

3.5 Non-Monotone Segmentation in the Discrete Trajectory Model

In this section we consider the segmentation problem for trajectories in the discrete model (see Section 1.2). The approach in which we compute a start-stop diagram and segment the trajectory based on the start-stop diagram works also in this case, and can even be simplified slightly.

We again segment the trajectory based on an attribute function f. Now $f\colon U \to \mathbb{R}$ is a discrete function defined on the index set $U = \{1,..,n\}$. A segmentation is a partition of U into disjoint (contiguous) subsequences, or *discrete segments*. Our task is now to compute a *discrete segmentation*: a segmentation into a minimum number of valid discrete segments. We will address the *weighted* version of the problem in which we assume that f is given as a sequence $S = s_1,..,s_n$ of pairs $s_i = (v_i, w_i)$, where v_i is the value of piece i and w_i is its weight. The weights can be used to represent pieces of different lengths.

Analogous to the start-stop diagram, we consider the *start-stop matrix*, a simplification of the start-stop diagram that is an $n \times n$ matrix B with boolean values, where the entry $B(i,j)$ at the ith column and jth row from below corresponds to the value of the criterion function at (i,j).[4] It is then relatively easy to compute a minimal segmentation by using dynamic programming, since we can use $B(\cdot,\cdot)$ as the adjacency matrix of an unweighted directed acyclic graph. A minimal segmentation corresponds precisely to a shortest path from 1 to n and can be found in $O(n^2)$ time [36].

▶ **Proposition 3.13.** *Given an $n \times n$ start-stop matrix, one can check if a discrete segmentation exists, and if so compute it, in $O(n^2)$ time and space.*

3.6 Computing the Start-stop Diagram

So far we focused on computing a minimal segmentation for a given start-stop diagram. We now discuss two specific criteria, prove that they are verticonvex, and explain how to construct the corresponding start-stop diagram efficiently. The criteria considered are the outlier-tolerant criterion and the standard-deviation criterion on a piecewise-constant function f. We also show that if f is piecewise linear then the start-stop diagram may contain more than one tunnel cell per row.

3.6.1 Segmentation on the Outlier-Tolerant Criterion

For a given piecewise-constant function f, we can segment f such that in each valid segment $[a,b]$ the minimum and maximum values differ by at most h. To accomodate outliers we require that only a fraction ρ, for a given constant $0 \leq \rho \leq 1$, of a valid segment $[a,b]$ must have its values within a range of

---

[4] Usually a matrix is indexed by row first and column second. We switch the indices to be consistent with the continuous case, and because it feels more natural in light of the application.

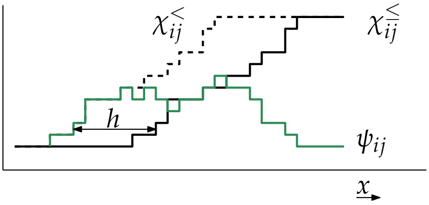


Figure 3.11: Obtaining the function ψ_{ij} from $\chi_{ij}^{\leq}$ and $\chi_{ij}^{<}$ shifted by h to the left (dashed).

extent h, and a fraction $1 - \rho$ of $[a, b]$ may have any value.[5] We present an efficient algorithm for computing the start-stop diagram, and further show that the forbidden space within each cell of the start-stop diagram is verticonvex. To simplify the presentation, we first present an algorithm to compute the start-stop matrix. That is, we consider discrete segmentation. We then extend this algorithm to compute the start-stop diagram.

Discrete Segmentation

We consider f as a sequence $s_1 = (v_1, w_1), \dots, s_n = (v_n, w_n)$ of (value, weight) pairs. We first give a formal definition of the criterion and investigate its structure. Let S_{ij} be the contiguous subsequence $s_i, \dots, s_j$ and let λ_{ij} be the total weight of its elements. Let $S_{ij}^{\leq}(x) = \{s_k \in S_{ij} \mid v_k \leq x\}$ denote the set of elements in S_{ij} with value at most x, and let $\chi_{ij}^{\leq}(x) = \sum_{s_k \in S_{ij}^{\leq}(x)} w_k$ be the total weight of these elements. We define $S_{ij}^{<}(x)$ and $\chi_{ij}^{<}(x)$ analogously. The total weight of the elements in S_{ij} with a value in the range $[x, x + h]$ is then

$$\psi_{ij}(x) = \chi_{ij}^{\leq}(x + h) - \chi_{ij}^{<}(x).$$

Our goal is to segment f so that, for each segment S_{ij}, we have $\max_x \psi_{ij}(x)/\lambda_{ij} \geq \rho$. Let B be the start-stop matrix. To test the value B_{ij}, we could compute an explicit representation of $\chi_{ij}^{\leq}$ and an explicit representation of $\chi_{ij}^{<}$ shifted by h to the left, and then take their difference (see Figure 3.11). This takes $O((j - i)\log(j - i))$ time, since we have to sort the values by weight. There are $O(n^2)$ cells, so the total amount of time required to compute the start-stop matrix is $O(n^3 \log n)$.

It is however not necessary to recompute the functions from scratch for each cell. Increasing j by one corresponds to raising the functions $\chi_{ij}^{\leq}$ and $\chi_{ij}^{<}$ by

---

[5] Using the same algorithm we can also handle the following similar criterion by using the logarithm of the function. For a given piecewise-constant positive function f and a ratio h, we can segment f so that in each valid segment $[a, b]$ there is a portion V of total length at least $\rho(b - a)$ such that $w_{\max}/w_{\min} \leq h$, where $w_{\min}$ and $w_{\max}$ are the minimum and maximum function values that occur in V.

w_{j+1} for all values $x \geq v_{j+1}$. Therefore we have:

$$\psi_{i(j+1)}(x) = \begin{cases} \psi_{ij}(x) + w_{j+1} & \text{if } x \in [v_{j+1} - h, v_{j+1}]\text{, and} \\ \psi_{ij}(x) & \text{otherwise.} \end{cases} \tag{3.6}$$

We now store ψ_{ij} as a data structure that allows us to query the maximum of ψ_{ij} on any given interval, and can be updated efficiently to represent $\psi_{i(j+1)}$. The data structure we use is an augmented segment tree that supports both operations in $O(\log n)$ time.[6]

A data structure to compute the start-stop matrix. We associate each piece $s_j = (v_j, w_j)$ of f with an interval $I_j = [v_j - h, v_j]$ of *weight* w_j. By Equation (3.6) it now follows that the value of $\psi_{ij}(x)$ is equal to the total weight of the intervals from $I_1, \ldots, I_j$ that contain x. Hence, we can represent ψ_{ij} using the set of intervals $I_1, \ldots, I_j$. We now describe an augmented segment tree T that stores these weighted intervals.

Let $u_1, \ldots, u_m$ be the endpoints of all intervals in $I_1, \ldots, I_n$ in sorted order. Internally, a segment tree T stores the elementary intervals $[u_i, u_{i+1}]$ in this order in the leaves of a balanced binary tree [37]. The internal nodes store values that allow searching on u-value. Since we know the endpoints of the intervals of ψ_{ij} for all i and j in advance, we can initialize T with the set of $O(n)$ endpoints and all weights set to zero. Therefore, no rebalancing has to be done when adding or removing an interval stored in the tree; only the values stored in the nodes which relate to weights will change.

Each node ν in a segment tree has an associated range r_ν, and an associated set $\mathcal{I}_\nu$ of intervals. The range r_ν is the union of the elementary intervals stored in (the leaves of) the subtree rooted at ν, and $\mathcal{I}_\nu$ is a subset of intervals stored in the tree. An interval I occurs in $\mathcal{I}_\nu$ if and only if I contains r_ν but not the range of ν's parent node [37]. We now augment T such that each node ν stores the total weight A_ν of the intervals associated with ν.

So for a tree T representing the function ψ_{ij} we can obtain the value of ψ_{ij} at x by searching for the elementary interval containing x and summing over the A-values on the search path.

A second augmentation provides us with a way to determine the maximum of ψ_{ij} in a given interval in $O(\log n)$ time. This is done by storing a value B_ν at every node ν that is the maximum sum of all A-values on a path from ν to a leaf in the subtree rooted at ν.

---

[6] The "segments" stored in this standard data structure as described in [37] are not to be confused with the segments of the segmentation of a trajectory.

When querying T for the maximum of the function ψ_{ij} on a given interval $[a, b]$, we walk along the two paths from the root to the elementary intervals containing a and b and maintain the maximum of the B-values stored in the roots of the subtrees between the two paths. This takes time linear in the number of nodes visited, hence we can find the maximum of ψ_{ij} on $[a, b]$ in $O(\log n)$ time.

When inserting an interval $I_j = [v_j - h, v_j]$ with weight w_j, updating A- and B-values can be done in $O(\log n)$ time. The A-value needs to be increased by w_j in the $O(\log n)$ nodes ν for which $I_j \in \mathcal{I}_\nu$. This operation is standard. The B-values have to be updated only in the nodes along the path from the root to the nodes where the A-values have been modified. Since those nodes lie along the two paths from the root to the elementary intervals storing $v_j - h$ and v_j, this can be done in $O(\log n)$ time overall.

▶ **Lemma 3.14.** *The start-stop matrix can be computed in $O(n^2 \log n)$ time.*

Proof. We fill in the matrix B by testing the validity of subsequences of S. A single column of B corresponds to testing the validity of $S_{ii}, S_{i(i+1)}, \dots, S_{in}$. This can be done in the given order (bottom-up in a column) in $O(n \log n)$ time by using the data structure described above.

Assume that we have a tree T representing ψ_{ij} and that we have determined whether S_{ij} is valid. Then we insert I_{j+1} with weight w_{j+1} in the augmented tree, and perform a query to determine $\max_x \psi_{i(j+1)}(x)$. We also compute $\lambda_{i(j+1)}$ from λ_{ij} by adding w_{j+1}. Now the test $\max_x \psi_{i(j+1)}(x)/\lambda_{i(j+1)} \geq \rho$ determines whether or not $S_{i(j+1)}$ is valid. ☐

Using the algorithm from Proposition 3.13 we can then conclude:

▶ **Proposition 3.15.** *Given a piecewise-constant function f with n breakpoints, a threshold value $h > 0$, and a ratio $\rho \in [0, 1]$, we can compute a discrete segmentation for the condition that, on a fraction of length at least ρ of a segment, the difference between the maximum and minimum function value is at most h, in $O(n^2 \log n)$ time.*

Continuous Segmentation

For continuous segmentation, we are allowed to cut f at any two points $a, b \in [0, 1]$. So we need to determine the forbidden space in the start-stop diagram. The breakpoints of the function f decompose the start-stop diagram into a grid. We now prove that the forbidden space within a cell $C = [\underline{a}, \overline{a}] \times [\underline{b}, \overline{b}]$ of this grid can be described by four linear inequalities of the following form, where

$\psi_1^*, \dots, \psi_4^*$ are constant values specific to the cell:

$$(b-a)\rho > \begin{cases} \psi_1^* + (\bar{a}-a) + (b-\underline{b}) \\ \psi_2^* + (\bar{a}-a) \\ \psi_3^* + (b-\underline{b}) \\ \psi_4^* \end{cases} \tag{3.7}$$

▶ **Lemma 3.16.** *For any cell $C = [\underline{a}, \bar{a}] \times [\underline{b}, \bar{b}]$ of the start-stop diagram, there exist values of $\psi_1^*, \dots, \psi_4^*$, such that the forbidden space in C is the intersection of four halfplanes within the cell, described by the inequalities in Equation (3.7).*

Proof. We use an approach similar to that used in the previous section. Let

$$\chi^{\leq}(x,a,b) = \left|\{t \mid t \in [a,b] \wedge f(t) \leq x\}\right|$$

denote the length of the portion of $[a,b]$ on which the value of f is at most x, let $\chi^{<}$ be defined analogously, and let

$$\psi(x,a,b) = \chi^{\leq}(x+h,a,b) - \chi^{<}(x,a,b). \tag{3.8}$$

For any point $(a,b) \in [\underline{a}, \bar{a}] \times [\underline{b}, \bar{b}]$, the corresponding segment $[a,b]$ is invalid if and only if $\max_x \psi(x,a,b) < (b-a)\rho$, i.e., for all possible intervals $I = [x, x+h]$, the fraction of $[a,b]$ on which the function value lies in I is smaller than ρ.

Note that the candidate segment corresponding to (a,b) contains the segment defined by $(\bar{a}, \underline{b})$. Let ψ_C be the function ψ restricted to the cell C. We can rewrite ψ_C with respect to $\psi(x, \bar{a}, \underline{b})$ as follows. Within the cell, decreasing a by some value Δ_a corresponds to raising the function $\chi^{\leq}$ by Δ_a for all input values which are greater than or equal to $f(a)$; the function $\chi^{<}$ is affected in a similar manner. By the definition of ψ in Equation (3.8) above, this implies that ψ_C increases by Δ_a only on the interval $I_a = [f(a)-h, f(a)]$. Similarly, increasing b by Δ_b results in increasing ψ_C by Δ_b on $I_b = [f(b)-h, f(b)]$. Letting $\Delta_a = \bar{a} - a$ and $\Delta_b = b - \underline{b}$, we can rewrite ψ_C as follows:

$$\psi_C(x,a,b) = \begin{cases} \psi(x,\bar{a},\underline{b}) + \Delta_a + \Delta_b & \text{if } x \in I_a \cap I_b, \\ \psi(x,\bar{a},\underline{b}) + \Delta_a & \text{if } x \in I_a \setminus I_b, \\ \psi(x,\bar{a},\underline{b}) + \Delta_b & \text{if } x \in I_b \setminus I_a, \\ \psi(x,\bar{a},\underline{b}) & \text{otherwise.} \end{cases} \tag{3.9}$$

Therefore

$$\max_x \psi_C(x,a,b) = \max \begin{cases} \max_{x \in I_a \cap I_b} \psi(x,\overline{a},\underline{b}) + \overline{a} - a + b - \underline{b} \\ \max_{x \in I_a \setminus I_b} \psi(x,\overline{a},\underline{b}) + \overline{a} - a \\ \max_{x \in I_b \setminus I_a} \psi(x,\overline{a},\underline{b}) + b - \underline{b} \\ \max_{x \notin (I_a \cup I_b)} \psi(x,\overline{a},\underline{b}), \end{cases} \tag{3.10}$$

so the condition $\max_x \psi(x,a,b) < (b-a)\rho$ is equivalent to the conjunction of four linear inequalities in a and b; hence the forbidden space within C is the intersection of the resulting four halfplanes. The four maxima on the right-hand side of Equation (3.10) are real numbers that depend only on $\overline{a}$ and $\underline{b}$; if we refer to them as $\psi_1^*, \ldots, \psi_4^*$, respectively, we obtain conditions of the form in Equation (3.7). ☐

▶ **Lemma 3.17.** *The start-stop diagram can be computed in* $O(n^2 \log n)$ *time.*

Proof. As in the discrete case, we can represent the function $\psi_{ij} = \psi(\cdot,\overline{a},\underline{b})$ in a data structure, where $(\overline{a},\underline{b})$ is the lower right corner of the current cell. Again, we maintain this data structure while traversing the cells of the grid bottom up within a column and reinitialize it for every column. We use exactly the same data structure as before. By Lemma 3.16, the forbidden space in a cell is described by Equation (3.7). We now need four queries to compute the values of $\psi_1^*, \ldots, \psi_4^*$, since these are the maxima of the function ψ_{ij} on the intervals in Equation (3.9). And hence, we get the free space of a cell C. This means we can compute the start-stop diagram in $O(n^2 \log n)$ time. ☐

Clearly, Lemma 3.16 implies the forbidden space within each cell of the start-stop diagram is verticonvex and has constant complexity, so we can invoke Theorem 3.3. We conclude:

▶ **Theorem 3.18.** *Given a piecewise-constant function* f *with* n *breakpoints, a threshold value* $h > 0$*, and a ratio* $\rho \in [0,1]$*, we can compute a minimal segmentation for the condition that on a fraction of at least* ρ *of a segment, the difference between the maximum and minimum function value is at most* h *in* $O(n^2 \log n + kn^2)$ *time, where* k *is the size of a minimal segmentation.*

Piecewise-linear functions. If the attribute function f is piecewise linear, then the grid induced by the breakpoints of f may contain (many) tunnel cells. Assume that we want to segment the attribute function f depicted in Figure 3.12 on the outlier-tolerant criterion with $h = 2+\varepsilon$, for some $0 \leq \varepsilon \leq 0.1$ and $\rho = 2/3$, i.e., the difference between any two values is not more than approximately 2 but we can disregard 1/3 of the segment. For this criterion, the candidate

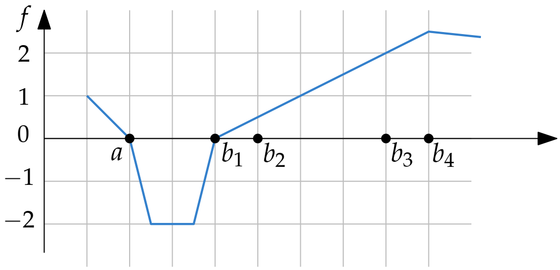


Figure 3.12: A piecewise-linear attribute function that leads to tunnel cells.

segments $[a, b_2]$ and $[a, b_3]$ are valid, as well as any candidate segment $[a, b]$ for $b_1 < b < b_2$. However, there exist candidate segments $[a, b]$ for $b_2 < b < b_3$ and for $b_3 < b < b_4$ which are not valid. Since b_1, b_2, b_3, and b_4 lie on the same piece of the function f, this implies that the vertical line at a intersects the forbidden space in this cell twice, once below b_3 and once above b_3.

It is now easy to see that we can create more than one tunnel cell per row: we simply use the above construction with points $a^- = a - \delta$ and $a^+ = a + \delta$, for some arbitrarily small δ, as starting point of the segments. Since a is a breakpoint, these points lie on different pieces of f, and hence we get two tunnel cells. In both cases the endpoints of the segments lie on the piece $[b_1, b_4]$, so the cells lie in the same row.

This example suggests that solving the problem for piecewise-linear attribute functions efficiently calls for a different approach.

3.6.2 Segmentation on the Standard-Deviation Criterion

Another important non-monotone criterion that we consider involves the standard deviation of an attribute function. In this section we show how to compute a minimal segmentation of a piecewise-constant function f where each segment has standard deviation not exceeding a given threshold value. Let $\mu(a, b) = \int_a^b f(y)\,\mathrm{d}y/(b - a)$ denote the mean value on a candidate segment $[a, b]$. The standard deviation $\sigma(a, b)$ is given by

$$\sigma(a, b) = \sqrt{\frac{\int_a^b (f(x) - \mu(a, b))^2\,\mathrm{d}x}{b - a}}\ .$$

Discrete Segmentation

When computing a discrete segmentation, we are allowed to partition f only where its value changes. Recall that f is given as a sequence $S = s_1, \ldots, s_n$ of pairs $s_i = (v_i, w_i)$, where v_i is the value of piece i and w_i is its weight.

▶ **Lemma 3.19.** *The start-stop matrix can be computed in $O(n^2)$ time.*

Proof. To compute the start-stop matrix B, we need to test whether the standard deviation of a segment is below the allowed threshold. We could compute this in time linear in the length of a segment. However, we can also maintain the mean μ_{ij} and standard deviation σ_{ij} of a weighted sequence S_{ij}. We can then compute the mean $\mu_{i(j+1)}$ and the standard deviation $\sigma_{i(j+1)}$ for $S_{i(j+1)}$ in constant time from μ_{ij} and σ_{ij}. This implies that we can fill B in constant time per cell and thus quadratic time in total. ☐

Once we have B we can compute a minimal segmentation in $O(n^2)$ time using the algorithm of Proposition 3.13. Hence:

▶ **Proposition 3.20.** *For any value $h > 0$, a discrete segmentation where each segment has standard deviation at most h can be computed in $O(n^2)$ time.*

Continuous Segmentation

In the continuous case, a and b can have any real value in $\mathbb{T}$. Setting the standard deviation $\sigma(a, b)$ equal to the constant threshold value h, we obtain the functional description of the boundaries of the forbidden space in the start-stop diagram:

$$\sigma(a,b) = h \iff \int\limits_a^b (f(x) - \mu(a,b))^2 \,\mathrm{d}x = (b-a)\cdot h^2.$$

Further algebraic manipulations, using the fact that f is piecewise constant, give a cubic expression in a and b. Hence, the boundaries of the forbidden space within each cell of the start-stop diagram are piecewise-cubic curves. This allows us to prove the following lemma:

▶ **Lemma 3.21.** *Every cell of the start-stop diagram is verticonvex.*

Proof. On a vertical line, the start point of the candidate segment is fixed. Let it be $\tilde{a}$. Within a single cell, the function f is constant. Hence, the function value $f(b) = c$ at the end point b of a segment $[\tilde{a}, b]$, for some $c \in \mathbb{R}$. Varying b implies including more or less of this constant c in the values whose standard deviation is considered.

Imagine b is at its low end of the cell, and let it increase to its high end. Then the mean $\mu(\tilde{a}, b)$ tends monotonically towards c. There are four cases to distinguish:

1. Initially $\sigma(\tilde{a}, b) < h$ and $|\mu(\tilde{a}, b) - c| < h$. Then the whole intersection of the cell and the vertical line is allowed.

2. Initially $\sigma(\tilde{a}, b) > h$ and $|\mu(\tilde{a}, b) - c| > h$. Then the lower end of the intersection is forbidden, but possibly, at some value of b it becomes allowed.

3. Initially $\sigma(\tilde{a}, b) < h$ and $|\mu(\tilde{a}, b) - c| > h$. Then the lower end of the intersection is allowed, but possibly, it becomes forbidden and possibly later allowed again.

4. Initially $\sigma(\tilde{a}, b) > h$ and $|\mu(\tilde{a}, b) - c| < h$. Then the lower end of the intersection is forbidden, but possibly, at some value of b it becomes allowed.

In other words, we have the property that if for some b the candidate segment becomes allowed, then it stays allowed. This is true for the following reason. At the value $\tilde{b}$ of b when $[\tilde{a}, b]$ becomes allowed, we have $\sigma(\tilde{a}, \tilde{b}) = h$ and $|\mu(\tilde{a}, \tilde{b}) - c| < h$. For increasing b the average $\mu(\tilde{a}, b)$ will tend monotonically towards c. So for larger b we have $|\mu(\tilde{a}, b) - c| < h$, and therefore $\sigma(\tilde{a}, b) < h$. This proves the lemma. ☐

▶ **Lemma 3.22.** *The start-stop diagram can be computed in $O(n^2)$ time.*

Proof. The forbidden space in each cell has constant complexity, and given the description of σ for cell C we can compute σ for neighbors of C in $O(1)$ time. ☐

Using Theorem 3.3, we then obtain:

▶ **Theorem 3.23.** *Given a piecewise-constant function f with n breakpoints and a threshold value $h > 0$, we can compute a minimal segmentation for the criterion that the standard deviation of a segment is at most h in $O(kn^2)$ time, where k is the size of a minimal segmentation.*

Piecewise-linear functions. As with the outlier-tolerant criterion, if f is piecewise linear then the grid decomposition of the start-stop diagram induced by the breakpoints of f may contain (many) tunnel cells. Consider for example the function f depicted in Figure 3.13. For both the candidate segments $[a, b_2]$ and $[a, b_4]$ the mean is close to 0.5 and the standard deviation is close to 0.25. However, there exists a value $b_2 < b < b_4$, such that the mean and the standard deviation of $[a, b]$ are smaller than 0.25. For example in Figure 3.13 the standard deviation at b_3 is below 0.2. So if we choose $h = 0.25$ the cell $[0, b_1] \times [b_1, b_5]$ is

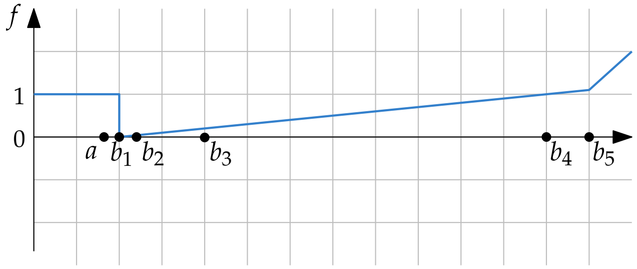


Figure 3.13: A piecewise-linear attribute function that leads to tunnel cells.

a tunnel cell: the vertical line at a intersects the forbidden space twice, once below b_3 and once above it. We can extend this example to construct two tunnel cells as before. Therefore, we do not expect that there exists a straightforward extension of our algorithm to this case.

3.7 Line Simplification

Even though we designed our framework for trajectory segmentation, the approach is also applicable to different problems. In this section, we discuss its use for certain polygonal line simplification problems.

Early line simplification algorithms try to reduce the number of vertices of a polygonal line while keeping its global shape by selecting a subset of the vertices [42, 73]. A different approach to the problem is taken by line simplification methods that allow the vertices of the simplified polyline to lie anywhere [65], and only require the shape to resemble the input curve. We propose a version of line simplification that compromises these two extremes: we restrict the vertices of the simplified polyline to lie on the input curve, but do not insist that they coincide with input vertices. A single edge of a simplified polyline is called a *shortcut* and it can replace a piece of the original polyline, namely the piece between the endpoints of the shortcut (which lie on the original polyline). We cast line simplification problems into our framework: a simplification is a staircase in the start-stop diagram.

The most common criterion for line simplification is the Hausdorff distance. An edge of a simplification with its endpoints on the input polyline is valid if the Hausdorff distance between that edge and the polyline piece it replaces is at most a pre-specified value. It is easy to verify that the resulting cells in the start-stop diagram can be tunnel cells, and there can be many in a single row. Hence, our approach does not give a polynomial-time algorithm for the simplification problem using the Hausdorff distance.

Next we consider the dilation criterion, where every edge of the simplification can be at most a given factor shorter than the polyline piece it replaces. Fix a starting point and an edge that contains the end point, and parameterize the position on the end edge. The function that gives the dilation expressed in this parameter is the ratio of a linear function and a hyperbolic function that does not grow faster than the linear function. It can easily be verified that the dilation measure gives verticonvex cells in the start-stop diagram.

Another criterion that we can handle easily is requiring that shortcuts cannot be shorter than a given threshold. This criterion also gives verticonvex cells in the start-stop diagram and we obtain the analogous result. However, the conjunction of the two criteria, dilation and non-shortness, does not give verticonvex cells, which is to be expected after the discussion in Section 3.3.

Now suppose that not only a polyline P but also a convex polygonal obstacle is given, and assume that they do not intersect. We are interested in the minimum-link simplification of P that also does not intersect the obstacle. It is easy to see that all cells are verticonvex. If the obstacle has m vertices, the complexity of the forbidden space in each cell is $O(m)$, and can be computed in $O(m)$ time by rotating a line along the boundary of the obstacle. Thus, we can compute the start-stop diagram in $O(n^2 m)$ time. When we compute the (outgoing) $(i+1)$-reachable intervals from the (incoming) i-reachable intervals in a cell C, we have to compute the highest (lowest) point p below (above) the forbidden space for which p_x lies in an incoming interval. Since all cells are verticonvex, the incoming intervals in C are incident to the left or right boundary of the cell, and they grow monotonically in each round. It follows that we can maintain p by walking along the boundary of the forbidden space. Therefore, we can compute the k-reachable space, where k is the length of a minimum-link simplification of P, in $O(n^2(m+k))$ time. The total running time of our algorithm is thus also $O(n^2(m+k))$.

Similarly, assume that a polyline in 3-dimensional space is given, together with a convex polyhedral obstacle. Again cells are verticonvex, and we obtain the same running time as in the planar case due to the following:

▶ **Lemma 3.24.** *The complexity of a cell in the start-stop diagram is $O(m)$, and can be computed in $O(m)$ time.*

Proof. For any cell C, let e and e' be the edges on which the starting and ending points lie, respectively. For any point a on e, consider the triangle $\triangle(a, e')$ that a forms with edge e', and consider how it intersects with the obstacle $\mathcal{O}$, see Figure 3.14. Because $\mathcal{O}$ is convex, there are up to two rays from a inside $\triangle(a, e')$ to e' that are tangent to $\mathcal{O}$, and generally these tangencies involve edges of $\mathcal{O}$. Now consider a moving over e from one endpoint to the other, and consider one such a tangent ray r. Let $b(a)$ be the point on e' hit by r. While $b(a)$ point moves,

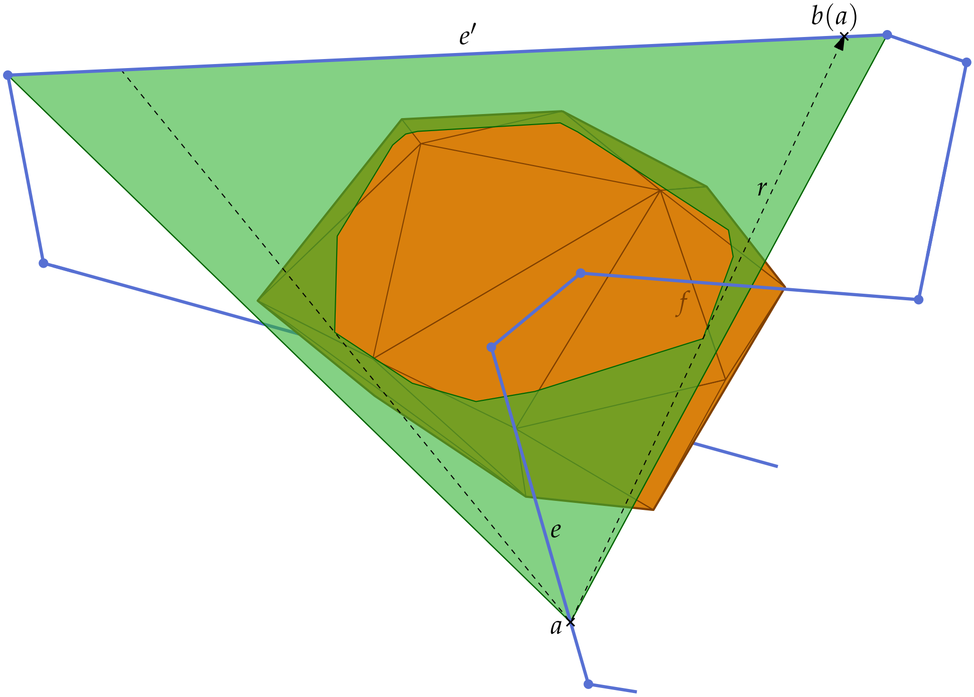


Figure 3.14: The triangle formed by p and e' intersecting an obstacle, and the tangents from p to e'.

it traces the boundary of the forbidden space in the cell C. As long as the edge f of $\mathcal{O}$ that defines r remains the same, the movement of $b(a)$ stays the same. Thus, we get one curve in C bounding the forbidden space. The edge f of $\mathcal{O}$ can change in two ways, causing a change in movement of $b(a)$, and thus a different curve bounding the forbidden space in C: (i) when r encounters a vertex, and (ii) when r aligns with a face of $\mathcal{O}$ incident to f. It is easy to see that a and $b(a)$ can become colinear with a vertex or a face of $\mathcal{O}$ at most once. It follows that the total number of events, and thus the complexity of the boundary of the forbidden space, is at most $O(m)$.

We can compute the boundary of the forbidden space by maintaining the obstacle edge f defining ray r, while moving a. We can determine the next event in $O(1)$ time. At an event of type (i) we find the new edge f' defining r in time proportional to the number of edges incident to the obstacle vertex. At an event of type (ii) this takes time proportional to the number of edges bounding the obstacle face. Since each vertex and each face generate at most one event, the total running time is $O(m)$. □

If the obstacle is not convex, then we can get tunnel cells and we do not obtain a polynomial-time algorithm. If we have more than one obstacle then we can get tunnel cells as well. We summarize our results in the following proposition.

▶ **Proposition 3.25.** *Given a polygonal line P with n vertices we can compute a minimum-link simplification P' of P where all k vertices of P' lie on P, and such that*

- *the dilation of each edge of P' is at most a given value δ, in $O(n^2k)$ time, if P lies in the plane;*
- *the length of each edge of P' is at least a given length γ, in $O(n^2k)$ time, if P lies in $\mathbb{R}^d$ (for any constant d); or*
- *the simplification P' does not intersect a given convex polyhedral obstacle $\mathcal{O}$, in $O(n^2(m+k))$ time, where m is the number of vertices in $\mathcal{O}$, if P and $\mathcal{O}$ lie in $\mathbb{R}^2$ or $\mathbb{R}^3$.*

3.8 Concluding Remarks

We analyzed the problem of segmenting a trajectory for non-monotone criteria. For monotone criteria efficient algorithms are known [31]. We showed that also for certain non-monotone criteria polynomial-time solutions exist. In particular, we presented a generic approach to segment a trajectory using a start-stop diagram, and we identified properties of the start-stop diagram that allow for efficient segmentation. Computing a segmentation from an arbitrary start-stop diagram is NP-hard.

For two concrete non-monotone criteria we presented efficient algorithms to compute the start-stop diagram. We also showed that the resulting diagrams indeed have the properties that allow for efficient segmentation. As a result, we can compute an optimal segmentation of a trajectory based on the outlier-tolerant criterion in $O(n^2 \log n + kn^2)$ time, and on the standard deviation criterion in $O(kn^2)$ time, where n is the number of vertices of the input trajectory and k is the number of segments in an optimal solution. These two criteria are relevant in practice, since they are more robust against noise than related monotone criteria.

The question for a complete characterization of the start-stop diagrams that allow for efficient segmentation is still open. We answered this question partially, but more can perhaps be said. Furthermore, the analyses of the running times for the outlier-tolerant criterion and the standard deviation criterion are based on the assumption that the attribute function is piecewise constant. While many second-order attributes, such as speed and heading, are piecewise constant due to the linear interpolation of the trajectory, it would also be interesting to see whether similar results can be obtained for, e.g., piecewise-linear functions. The examples given in Sections 3.6.1 and 3.6.2 seem to indicate that the methods developed in this paper do not easily apply if the attribute functions are piecewise linear.

Finally, our two-step approach of first computing the start-stop diagram explicitly, and then solving the segmentation problem, inherently requires at least $\Omega(n^2)$ time and space. An intriguing open problem is whether a subquadratic solution may be possible.

Chapter 4

Finding Hotspots

In this chapter we focus on the detection of interesting places in trajectory data. In particular, we give algorithms to identify regions where a moving entity spends a large amount of time. We refer to such a region as a *hotspot*, and model it as an axis-aligned square whose location is not known yet. We distinguish several versions of the problem of finding square-shaped hotspots. The problems we consider are:

1. The size of the square is fixed and we wish to find the placement that maximizes the time the entity spent inside. Here we allow the entity to leave the region and return to it later; all visits count for the duration.
2. We are given a duration and want to determine the smallest square and its placement so that the entity is inside for at least the given duration.
3. We consider problem 1, but now we are interested in *contiguous* presence inside the square, so only one of the visits to the square counts.
4. The same for problem 2.
5. We do not fix duration nor square size, but optimize a relative measure that is the ratio of the duration and the square side length.
6. We consider optimizing a relative measure like in problem 5, but now for contiguous presence in the square.

We present our results for a single trajectory, but they immediately extend to the case of multiple trajectories.

Problem statement. Recall that a trajectory $\mathcal{T}$ is a continuous piecewise-linear function, mapping time to points in the plane, that we write $\mathcal{T}(t)$ for the point on $\mathcal{T}$ at time $t \in \mathbb{T}$, and that we denote the *sub-trajectory* from time s to time t, with $s \leq t \in \mathbb{T}$, by $\mathcal{T}[s,t]$. The edges of $\mathcal{T}$ are line segments in the plane. With slight abuse of notation we use $\mathcal{T}$ to denote the trajectory as well as the set of edges $\{e_1, .., e_n\}$ of the trajectory. An edge e between points u and v, is a line segment $e = \overline{uv}$ of length $\|e\|$. We use the same notation to denote the length of (sub-)trajectories. By general position, we assume that the end points of these segments, the vertices of $\mathcal{T}$, all have distinct x-coordinates and y-coordinates. Our results do not depend on this assumption but the description is considerably easier because of it.

Let $\mathcal{H} \subset \mathbb{R}^2$ denote the axis-parallel square hotspot with center c and side length $2r$. We refer to r as the *radius* of $\mathcal{H}$. The boundary of $\mathcal{H}$ is denoted by $\partial\mathcal{H}$. The function $\Xi(\mathcal{H}) = \sum_{e \in \mathcal{T}} \|e \cap \mathcal{H}\|$ describes the total length of the trajectory inside $\mathcal{H}$. Similarly, $\Phi(\mathcal{H}) = \max\{\|\mathcal{T}[s,t]\| \mid s,t \in \mathbb{T} \wedge \mathcal{T}[s,t] \subseteq \mathcal{H}\}$ denotes the length of the longest (contiguous) sub-trajectory in $\mathcal{H}$. We define $\Psi(\mathcal{H}) = \Xi(\mathcal{H})/2r$ and $\Gamma(\mathcal{H}) = \Phi(\mathcal{H})/2r$ to denote the *relative* total trajectory length, and the *relative* contiguous trajectory length, in $\mathcal{H}$, respectively. If the center c and/or the radius r of a hotspot $\mathcal{H}$ are clear from the context, we may sometimes write $\Xi(c,r)$, $\Xi(c)$, or $\Xi(r)$ instead of $\Xi(\mathcal{H})$. We do the same for Φ, Ψ, and Γ. We can now formalize the five problems that we study as follows.

1. Given r, maximize $\Xi(\cdot, r)$ over all square placements.
2. Given L, find a smallest hotspot $\mathcal{H}^*$ with $\Xi(\mathcal{H}^*) \geq L$ over all square placements and sizes.
3. Given r, maximize $\Phi(\cdot, r)$ over all square placements.
4. Given L, find a smallest hotspot $\mathcal{H}^*$ with $\Phi(\mathcal{H}^*) \geq L$, over all square placements and sizes.
5. Find a hotspot $\mathcal{H}^*$ that maximizes Ψ, over all square placements and sizes.
6. Find a hotspot $\mathcal{H}^*$ that maximizes Γ, over all square placements and sizes.

We describe our algorithms by considering length rather than duration because it is more intuitive and easier to describe. Adapting the algorithms to consider duration is straightforward, however. We simply observe that all our results still hold for *weighted* edges, and that we can use the duration to assign a weight to each edge. We then simply choose the weights so that maximizing weighted length inside $\mathcal{H}$ is the same as maximizing duration inside $\mathcal{H}$.

Results and organization. Our algorithms that solve the above problems all use a key property of Ξ, namely that it is a linear function in c and r. We prove this in Section 4.1, and discuss how the shape of $\mathcal{H}$ affects this property. In Section 4.2 we study problems 1 and 2, and show that we can solve them in $O(n^2)$ time and $O(n^2 \log^2 n)$ time, respectively. We focus on the contiguous-length variants, problems 3 and 4, in Section 4.3. We show that we can solve both problems in $O(n \log n)$ time. In Section 4.4 we present an $O(n^3)$ time algorithm to maximize Ψ, and an $O(n^2)$ time algorithm to maximize Γ in $\mathcal{H}$. That is, we solve the relative length versions of the problem: problems 5 and 6. We review some extensions, like multiple entities and different shapes for $\mathcal{H}$, in Section 4.5.

4.1 Properties of Ξ

We start by showing that Ξ and Φ are piecewise-linear functions, which is key in our algorithms.

▶ **Lemma 4.1.** *Consider a square hotspot $\mathcal{H}$ with center c and radius r. The function Ξ is piecewise linear in c and r. The break points of Ξ correspond to hotspots such that: (i) a vertex of $\mathcal{T}$ lies on a side of $\mathcal{H}$, or (ii) a corner of $\mathcal{H}$ lies on an edge of $\mathcal{T}$.*

Proof. Fix an edge $e \in \mathcal{T}$, and consider the function $\Xi_e(\mathcal{H}) = \|e \cap \mathcal{H}\|$, denoting the length of the part of e that lies in $\mathcal{H}$. Next, we show that while e intersects a fixed set of sides of $\mathcal{H}$, the function Ξ_e is linear (in c and r). It then follows that Ξ_e is piecewise linear, and has a break point when one of the vertices of e intersects the boundary of $\mathcal{H}$, or when e intersects a corner of $\mathcal{H}$. Since Ξ is the sum of these functions Ξ_e, for $e \in \mathcal{T}$, it is also piecewise linear, and has a break point when one of the functions Ξ_e has a break point. Hence, the break points of Ξ correspond exactly to events of type (i) and (ii).

Since $\mathcal{H}$ is convex, the intersection between $e = \overline{uv}$ and $\mathcal{H}$ is single contiguous line segment, a singleton point, or it is empty. Fix the set of sides of $\mathcal{H}$ intersected by e, let $p_\lambda = (1-\lambda)u + \lambda v$, and let α and β be functions such that $\overline{p_{\alpha(c,r)}p_{\beta(c,r)}} = e \cap \mathcal{H}$. We then have $\Xi_e(\mathcal{H}) = \Xi_e(c,r) = \|e \cap \mathcal{H}\| = \|e\| \left(\beta(c,r) - \alpha(c,r)\right)$. It is now an easy exercise to show that α and β are linear functions in c and r. It follows that $\|e \cap \mathcal{H}\|$ is piecewise linear in c and r, and hence Ξ is piecewise linear as well. ☐

Let Ξ_π denote the function Ξ, restricted to a single piece π. Since the sum of a set of linear functions is linear, there is a constant-size description of Ξ_π. It follows that we can evaluate Ξ_π for some $\mathcal{H} \in \pi$, and compute the derivative of Ξ_π in constant time.

Using essentially the same argument as in Lemma 4.1 we can show that Φ is piecewise linear. We characterize the break points of Φ in Section 4.3.

Hotspots with curved boundaries. When $\mathcal{H}$ has a curved boundary, we can no longer describe the intersection points between $\partial\mathcal{H}$ and a single edge using linear functions of c and r. For example, in case $\mathcal{H}$ is a disk, the intersection points are described by an equation of the form $\sqrt{\gamma(c,r)}$, where γ is a quadratic function in c and r. This means that on a single piece, Ξ is the sum of square roots. A description of this function has linear size, and linear time is required to evaluate it. Worse still is that we can no longer analytically compute the roots of the derivative of Ξ. Since we need those roots to compute the maximum of Ξ, we can no longer maximize Ξ either. For this reason, we focus on square hotspots.

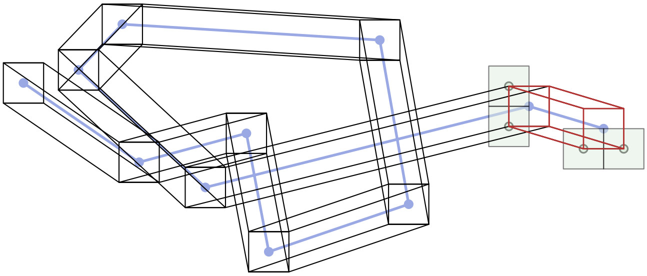


Figure 4.1: A trajectory and the corresponding subdivision $\mathcal{S}$. The line segments corresponding to a single edge e are shown in red.

4.2 Total Length

4.2.1 Fixed Radius

When the radius r of $\mathcal{H}$ is fixed, we can compute a placement that maximizes Ξ using a method similar to that of Mount, Silverman, and Wu [98]. Let c be the center of $\mathcal{H}$, and consider the parameter space of c. We compute a subdivision $\mathcal{S}$ of the parameter space such that inside each cell, Ξ is a simple linear function. It follows that maxima occur only at vertices of this subdivision. For each cell C, we compute a description of Ξ and evaluate it at the vertices of C. We can then just select the maximum over all cells.

Each edge of $\mathcal{T}$ yields $O(1)$ line segments in $\mathcal{S}$ (see Figure 4.1). Thus $\mathcal{S}$ is the arrangement of $O(n)$ line segments and can be constructed in $O(n^2)$ time.

Let Ξ_C denote the function Ξ restricted to cell C. In each cell of $\mathcal{S}$ we can compute Ξ_C and its maximum from scratch. This takes $O(n)$ time. However, for neighboring cells C and D, Ξ_C and Ξ_D can differ in only two ways: (i) the set of contributing edges has increased or decreased by one or two edges, or (ii) an edge intersects a different side of $\mathcal{H}$ (Lemma 4.1). So, we can compute a function $\Delta_{C,D}$ that describes these changes in constant time. We then have $\Xi_D(c) = \Xi_C(c) + \Delta_{C,D}(c)$, and thus we can compute Ξ_D from Ξ_C in constant time. Since $\mathcal{S}$ consists of $O(n^2)$ cells we conclude:

▶ **Theorem 4.2.** *Given a radius r, we can find a hotspot $\mathcal{H}^*$ with radius r that maximizes Ξ in $O(n^2)$ time.*

4.2.2 Fixed Length

We are given a threshold L, and we need to find the center c^* and the radius r^* of a minimum-size hotspot $\mathcal{H}^*$ with $\Xi(c^*, r^*) \geq L$.

The general idea is to use Megiddo's parametric search technique that we described in Chapter 2.1. We can use our algorithm from the previous section as the sequential decision algorithm $\mathcal{A}$, by simply computing the maximum length L^* in a hotspot of radius r and checking if $L^* \geq L$. So all that remains is to construct an analogous parallel decision algorithm. This will give us an efficient (sequential) algorithm to compute r^*.

A parallel decision algorithm. Let r be the given radius. We use the same approach as in the previous section: we construct the subdivision $\mathcal{S}$, and traverse all cells C to compute Ξ_C and its maximum. Constructing $\mathcal{S}$ and its dual graph $\mathcal{G}$ takes $O(\log\log^* n)$ time, using $O(n^2/\log n)$ processors [10]. The graph $\mathcal{G}$ has a node v for every face F_v of $\mathcal{S}$, and an arc (u, v) for each pair of adjacent faces F_u and F_v.[1]

Next, we compute a spanning tree of $\mathcal{G}$ and its Euler tour $\mathcal{E} = \epsilon_1, .., \epsilon_m$, with $m = O(n^2)$. This takes $O(\log n)$ time using $O(n^2)$ processors [104, 111]. Let Ξ_i denote the function Ξ in node ϵ_i, and let $\Delta_{i,j}$ be the difference function between Ξ_i and Ξ_j, that is, $\Delta_{i,j}(c) = \Xi_j(c) - \Xi_i(c)$. For two consecutive nodes ϵ_i and ϵ_{i+1} we can compute this function $\Delta_{i,i+1}$ without having computed Ξ_i and Ξ_{i+1} since we know which trajectory edge starts or stops to intersect $\mathcal{H}$ (or which edge now intersects a different side of $\mathcal{H}$). Once we have $\Delta_{i,j}$ and $\Delta_{j,\ell}$ we can obtain $\Delta_{i,\ell}$ by combining the two functions. So, the main idea is to compute function Ξ_1, and all functions $\Delta_{1,i}$ to obtain the functions Ξ_i. Next, we show that this can be done using $O(n^2)$ processors in $O(\log n)$ time.

▶ **Lemma 4.3.** *Given $O(n)$ processors, we can compute the function Ξ for a given node v of $\mathcal{G}$ in $O(\log n)$ time.*

Proof. We use one processor for each trajectory edge e. Each processor computes the function $\Xi_{ev}(c) = \|e \cap \mathcal{H}\|$. We then add these functions together in pairs of two, and repeat this process until we have one function representing Ξ. This takes $O(\log n)$ steps. ☐

▶ **Lemma 4.4.** *Given $O(m)$ processors, with $m \geq n$, and a path $\mathcal{E}$ of length m, we can compute all functions Ξ_i in $O(\log m)$ time.*

Proof. By Lemma 4.3 we can compute Ξ_1 in $O(\log n) = O(\log m)$ time. We now show how to compute all functions $\Delta_{1,i}$ in $O(\log m)$ time in total. We then use one processor for each i to compute Ξ_i from Ξ_1 and $\Delta_{1,i}$ in constant time.

We represent $\mathcal{E}$ as a balanced binary tree T. Each node $v_{i,j}$ in T represents a subpath $\epsilon_i, .., \epsilon_j$. The leaves of T simply correspond to the singleton paths ϵ_i. We compute the functions $\Delta_{i,j}$ for each internal node in the tree using a bottom-up

---

[1] To avoid confusion with the vertices and edges of $\mathcal{S}$, we use the terms "node" and "arc" for the vertices and edges of graph $\mathcal{G}$.

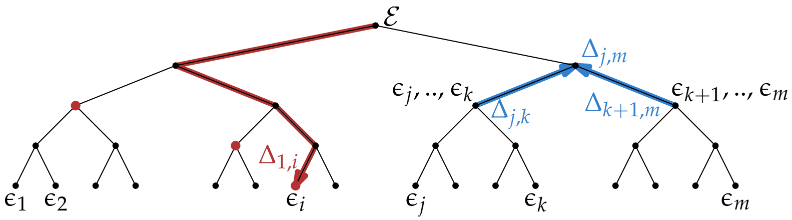


Figure 4.2: We compute $\Delta_{i,j}$ for all nodes in the tree bottom up (blue arrows), then we compute all $\Delta_{1,i}$ for all leaves ϵ_i by traversing a root to leaf path, and summing the red nodes.

procedure. With $O(m)$ processors, each one starting in a different leaf, this takes $O(\log m)$ time. For a given leaf ϵ_i, we can compute $\Delta_{1,i}$ by traversing T from the root to ϵ_i: we sum the functions stored at the left child of each node v where the path turns right, and add this to the function stored at the leaf of ϵ_i (see Figure 4.2). This takes $O(\log m)$ time. Since we have $O(m)$ processors, we can compute all functions $\Delta_{1,i}$ in parallel. □

By Lemma 4.4 we can compute Ξ_i for all nodes in $\mathcal{E}$ in $O(\log m) = O(\log(n^2)) = O(\log n)$ time using $O(n^2)$ processors. Once we have all functions Ξ_i, we can use the same tactic to compute the global maximum in $O(\log n)$ time.

Proposition 4.5. *Given a threshold L, a radius r, we can decide if there is a hotspot $\mathcal{H}$ with center c and radius r such that $\Xi(c, r) \geq L$ in $O(\log n)$ time using $O(n^2)$ processors.*

Computing a minimum-size hotspot. We use the above decision algorithm together with the parametric search technique. This gives us a sequential algorithm to compute a minimum-size hotspot. In general parametric search takes $O(PT_P + T_P T \log P)$ time (Theorem 2.2). From Theorem 4.2 we have $T = O(n^2)$, and from Proposition 4.5 we have $T_P = O(\log n)$ and $P = O(n^2)$. Plugging in these results gives us:

Theorem 4.6. *Given a threshold L, we can find a minimum-size hotspot $\mathcal{H}^*$ with center c^* and radius r^* such that $\Xi(c^*, r^*) \geq L$ in $O(n^2 \log^2 n)$ time.*

4.3 Contiguous Length

In this section we focus on finding a hotspot containing a longest contiguous sub-trajectory. Recall that $\mathcal{T}[s, t]$ is the sub-trajectory between s and t. With some abuse of notation we will also write $\mathcal{T}[p, q] = \mathcal{T}[t_p, t_q]$ for the sub-trajectory between points $p = \mathcal{T}(t_p)$ and $q = \mathcal{T}(t_q)$.

4.3.1 Fixed Radius

Let $\mathcal{T}^* = \mathcal{T}[p^*, q^*]$ be a longest sub-trajectory contained in any hotspot of radius r, and let $w = 2r$. We will state and prove properties of $\mathcal{T}^*$ that will help us to compute it efficiently. First we make the simple observation that the starting point p^* of $\mathcal{T}^*$ will be on the boundary of $\mathcal{H}^*$, with the one exception when $\mathcal{T}^*$ starts at the start of the trajectory $\mathcal{T}$. This exception is easy to handle in $O(n)$ time, so we ignore it and assume that $\mathcal{T}^*$ starts at a point p^* on $\partial\mathcal{H}^*$.

▶ **Lemma 4.7.** *There is a hotspot $\mathcal{H}^*$ that maximizes Φ, and such that at least one vertex of $\mathcal{T}$ lies on the boundary of $\mathcal{H}^*$. Vertex v lies on $\mathcal{T}^*$.*

Proof. Proof by contradiction. Assume that $\mathcal{H}^*$ maximizes Φ, and there is no hotspot $\mathcal{H}$ with $\Phi(\mathcal{H}) \geq \Phi(\mathcal{H}^*)$ with a vertex on $\partial\mathcal{H}$. Since $\mathcal{H}^*$ is optimal, the longest contiguous sub-trajectory $\mathcal{T}^* = \mathcal{T}[p^*, q^*]$ in $\mathcal{H}^*$ must touch two opposite sides of $\mathcal{H}^*$. Assume without loss of generality that these sides are horizontal.

Let $\mathcal{H}' = \mathcal{H}^*$ and let $\mathcal{T}'$ be a maximal length sub-trajectory in $\mathcal{H}'$, such that initially $\mathcal{T}' = \mathcal{T}^*$. It is easy to see that we can shift $\mathcal{H}'$ horizontally—while keeping $\mathcal{T}^*$ inside it—until either a vertex of $\mathcal{T}'$ lies on (a vertical side of) $\partial\mathcal{H}'$ or p^* lies on a corner of $\mathcal{H}'$. In the former case we immediately obtain a contradiction. In the latter case, translate $\mathcal{H}'$ while keeping the starting point p' of $\mathcal{T}'$ on the same corner of $\mathcal{H}'$ (moving the starting point of $\mathcal{T}'$ earlier or later). Let $\phi(t)$ denote the length of $\mathcal{T}'$ as a function of the starting time $t = t_{p'}$ of $\mathcal{T}'$. Function ϕ has break points when p' or q' crosses a vertex or when $\mathcal{H}'$ gets a vertex of $\mathcal{T}'$ on its boundary. Since ϕ is (piecewise) linear, there is at least one direction such that we can translate $\mathcal{H}'$ without decreasing $\phi(\mathcal{H}')$ until ϕ is at a break point. At such a break point $\mathcal{H}'$ has a vertex of $\mathcal{T}'$ on its boundary. Contradiction. This completes the proof. ☐

▶ **Corollary 4.8.** *For starting point $p^* = (p^*_x, p^*_y)$ of $\mathcal{T}^*$, and some vertex $v = (v_x, v_y)$ of $\mathcal{T}$ we have that (i) $p^*_x \in \{v_x - w, v_x, v_x + w\}$ or $p^*_y \in \{v_y - w, v_y, v_y + w\}$, and (ii) $v \in \mathcal{T}^*$.*

We now examine all vertices and all six cases from Corollary 4.8. We know that one of these cases will give us the starting point p^* of $\mathcal{T}^*$.

Let v be some vertex of $\mathcal{T}$ and assume that we are testing the case where $p^*_y = v_y - w$; all other cases are symmetric or handled analogously. In this case, the unknown point p^* lies on the bottom side of $\mathcal{H}^*$ and v lies on the top side. Furthermore, t_{p^*} is the earliest time for which $p^*_y = v_y - w$, and $\mathcal{T}[p^*, v] \subseteq \ell^+_{p^*}$, where $\ell^+_{p^*}$ is the upper half-plane bounded by the horizontal line with y-coordinate $p^*_y = v_y - w$. Assume that we have the earliest point

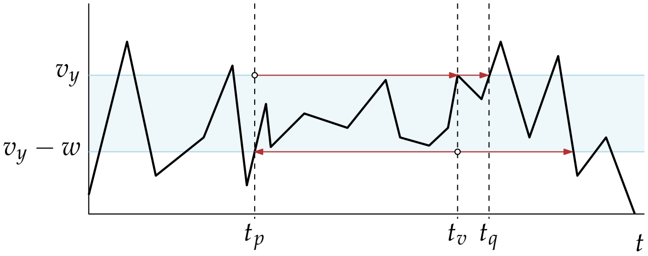


Figure 4.3: Determining the earliest starting time t_p, and the latest ending time t_q such that $\mathcal{T}[p,q]$ lies in the slab $[v_y - w, v_y]$.

p before v with y-coordinate $v_y - w$, and such that $\mathcal{T}[p,v] \subseteq \ell_p^+$. If $\mathcal{T}[t_p,t_v]$ contains any point with y-coordinate greater than v_y, then p is not a candidate for the start of the optimal sub-trajectory $\mathcal{T}^*$, otherwise we proceed to compute a maximal time interval I starting at p such that I has its y-coordinates in the range $[v_y - w, v_y]$. Suppose that we have $I = [p,q]$. Then we test whether I satisfies the condition that the x-extent of $\mathcal{T}[p,q]$ is at most w. If so, then $\mathcal{T}[p,q]$ is a candidate to be $\mathcal{T}^*$. After testing all vertices v and all six cases, we choose the longest one as $\mathcal{T}^*$.

We need to find an efficient way to implement the query for p, the query for q, and the test whether $\mathcal{T}[p,q]$ has an x-extent of at most w. Consider the polygonal line representing $\mathcal{T}(t)_y$ as a function of t, see Figure 4.3. Assume that we have a horizontal decomposition of this polyline, preprocessed for efficient planar point location [37]. This means that we can perform horizontal ray shooting queries in $O(\log n)$ time. A horizontal ray shooting query to the left from a point $(\hat{t},\hat{y})$ asks for the most recent time before $\hat{t}$ where the trajectory has y-coordinate $\hat{y}$. Similarly, a ray shooting query to the right asks for the earliest time after $\hat{t}$ where the trajectory has y-coordinate $\hat{y}$. Hence, to determine p given v, we perform a ray shooting query to the left from the point $(t_v, v_y - w)$, giving us $(t_{p'}, p'_y)$. By construction, we have $\mathcal{T}[p',v] \subset \ell_{p'}^+$, so either the trajectory enters the vertical slab $[v_y - w, v_y]$ at p', and thus $p = p'$, or p' is a vertex where the y-coordinate of $\mathcal{T}$ is at a local minimum. By our general position assumption there is at most one such vertex, so with one more ray shooting query to the left we find p.[2]

To find point q, we then perform a ray shooting query to the right from $(t_p, p_y + w) = (t_p, v_y)$. If the ray hits an edge before t_v then p is not a candidate start point and we stop. If the ray hits v and $\mathcal{T}$ extends upwards after v, then $[t_p, t_v]$ is a candidate time interval that could give $\mathcal{T}^*$. Otherwise, v is a local maximum and we perform two ray shooting queries to the right: from (t_v, v_y) and from $(t_v, v_y - w)$. The earliest time of the two answers gives us q and thus

---

[2] We can handle degenerate situations where there may be a sequence of vertices all with x-coordinate $v_y - w$ by considering the line segments to be open-ended at these vertices.

the interval I to test.

Finally, we test whether the x-extent of I is at most w. This can be done in constant time, using a data structure storing the x-coordinates of the vertices of $\mathcal{T}$ (in order along the trajectory) and supporting range minimum (and range maximum) queries. Such a data structure can be built in $O(n)$ time [15]. Alternatively, we can answer these queries in $O(\log n)$ query time, after $O(n \log n)$ preprocessing time using an augmented binary search tree: every leaf represents an edge of $\mathcal{T}$, and every internal node represents a sub-trajectory. Every internal node is augmented with two values: the minimum and maximum occurring x-coordinate of the edges in the leaves below it.

Analysis. In the preprocessing we build the ray shooting structures (for (t, y) and (t, x)) and the data structures to test the x-extent of an interval (for x and for y). This takes $O(n \log n)$ time. Then for every vertex v, we can find candidate intervals in $O(\log n)$ time.

▶ **Theorem 4.9.** *Given a radius r, we can find a hotspot $\mathcal{H}^*$ with radius r that maximizes Φ in $O(n \log n)$ time.*

4.3.2 Fixed Length

We are given a threshold L on the minimum required trajectory length in the hotspot, and we want to find a smallest hotspot $\mathcal{H}^*$ that contains a sub-trajectory of length L.

Let $\phi(t)$ be the minimum radius of a hotspot $\mathcal{H}$ containing a sub-trajectory $\mathcal{T}[p, q]$ of length L that enters $\mathcal{H}$, and thus starts, at time $t = t_p$, and let u and v be the first and the last internal vertex in $\mathcal{T}[p, q]$, respectively. We denote the bounding box of a sub-trajectory $\mathcal{T}[a, b]$ by $\mathcal{BB}(a, b)$. We now prove:

▶ **Lemma 4.10.** *The function ϕ is piecewise linear. The $O(n)$ break points of ϕ correspond to hotspots $\mathcal{H}$ such that: (i) p is a vertex of $\mathcal{T}$, (ii) q is a vertex of $\mathcal{T}$, (iii) p_x (p_y) coincides with the minimum or maximum x-coordinate (y-coordinate) of $\mathcal{BB}(u, v)$, (iv) q_x (q_y) coincides with the minimum or maximum x-coordinate (y-coordinate) of $\mathcal{BB}(u, v)$, and (v) $p_x = q_x$ or $p_y = q_y$.*

Proof. We start by showing that ϕ is piecewise-linear. Let e be the edge containing p and f the edge containing q. If we move point p along e by some small amount Δ, the length inside the hotspot decreases linearly in Δ. To maintain length L inside $\mathcal{H}$, point q has to move along f (see Figure 4.4). The required increase in length on f is identical to the decrease in length on edge e, namely Δ. It follows that p and q both move linearly in Δ. Therefore, the radius of $\mathcal{H}$ is linear as well. This in turn implies that ϕ is linear.

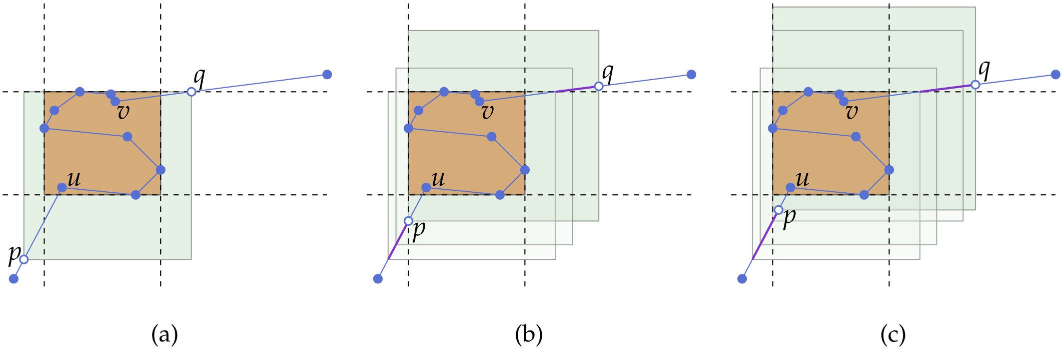


Figure 4.4: The smallest hotspot for several different starting times. The offset in length Δ is indicated in purple, and $\mathcal{BB}(u, v)$ in orange. (a) and (b) correspond to break points of a piece i of ϕ. (c) is a hotspot on piece $i+1$.

Next, we show that the break points of ϕ are of types (i) to (v). At any time t, a hotspot $\mathcal{H}$ with radius $\phi(t)$ has two opposite sides, s_1 and s_2, that both contain an internal vertex or an end point of $\mathcal{T}[p, q]$. As time t varies, s_1 and s_2 move. Function ϕ has a break point if and only if the movement of s_1 and s_2 changes. This movement changes when the movement of p and q changes, and when the objects defining s_1 and s_2 change (e.g. s_1 was defined by p, but is now defined by an internal vertex of $\mathcal{T}[p, q]$). The former changes occur when p and q are at trajectory vertices, thus yielding break points of type (i) and (ii). The latter changes occur exactly at events of type (iii) to (v), thus yielding break points of type (iii) to (v).

Finally, we argue that there are $O(n)$ break points. Clearly, there are $O(n)$ break points of type (i) and (ii). It follows that there are also only $O(n)$ pairs of edges (e, f) such that p lies on e and q on f. For each such a pair there are at most $O(1)$ break points of type (v). Point p (q) encounters at most $O(1)$ events of type (iii) (type (iv)) per edge. So the number of break points of these types is $O(n)$ as well. □

Note that ϕ is a partial function. In particular, ϕ is not defined for times t such that $\mathcal{T}[t]$ lies in the interior of $\mathcal{BB}(u, v)$, where u and v are the first and last interior vertices in sub-trajectory of length L starting at time t.

Since ϕ is piecewise linear, its minimum occurs at a break point. So, to find a smallest hotspot containing length L, we compute all break points of ϕ and evaluate ϕ at each of them.

We can easily find all break points of ϕ by "sweeping" $\mathcal{T}$ with a sub-trajectory $\mathcal{T}[p, q]$ of length L. To quickly find the bounding box and the length of a sub-trajectory we represent $\mathcal{T}$ as a balanced binary search tree. Each leaf node represents a trajectory edge e, and stores e, its bounding box, and its length. An internal node $\nu_{a,b}$ represents the sub-trajectory $\mathcal{T}[a,b]$ from vertex a to vertex b, and stores $\mathcal{BB}(a,b)$ and the length of $\mathcal{T}[a,b]$. Building this tree takes $O(n \log n)$ time, and allows $O(\log n)$ time queries. Once we have the bounding box and the length of $\mathcal{T}[u,v]$, we can construct and evaluate ϕ in constant time. We have $O(n)$ events in total, each of which we can handle in $O(\log n)$ time. Hence, we can find the global minimum in $O(n \log n)$ time. Thus:

▶ **Theorem 4.11.** *Given a threshold L, we can find a minimum-size hotspot $\mathcal{H}^*$ such that $\Phi(\mathcal{H}^*) \geq L$ in $O(n \log n)$ time.*

4.4 Relative Length

4.4.1 Total Length

We now focus on finding a hotspot $\mathcal{H}^*$ with center c^* and radius r^* that maximizes the relative trajectory length $\Psi(\mathcal{H}^*) = \Psi(c^*, r^*) = \Xi(c^*, r^*)/2r^*$.

Given a hotspot $\mathcal{H}$, a point $p \in \mathcal{H}$, and a radius r. Let $\mathcal{H}_p^r$ be the hotspot $\mathcal{H}$, scaled with p as origin and such that its radius is r. Fix a point p, and consider Ψ as a function of r. More formally, let $\psi_p(r) = \Psi(\mathcal{H}_p^r)$.

▶ **Lemma 4.12.** *ψ_p is a piecewise-hyperbolic function. The pieces of ψ_p are of the form $c(1/r) + d$, for $c, d \in \mathbb{R}$, and the break points of ψ_p correspond to hotspots $\mathcal{H}$ such that: (i) a vertex of $\mathcal{T}$ lies on a side of $\mathcal{H}$, or (ii) a corner of $\mathcal{H}$ lies on an edge of $\mathcal{T}$.*

Proof. It is easy to see that the break points of ψ_p are the same as those of Ξ. We now show that each piece of ψ_p is of the form $c(1/r) + d$, with $c, d \in \mathbb{R}$. Consider a piece of ψ_p, and let E be the set of contributing edges on that piece. Let $A \subseteq E$ be the set of edges that are completely contained in any hotspot corresponding to this piece, and let $B = E \setminus A$ be the set of remaining edges. For each edge e in B the length in $\mathcal{H}_p^r$ changes linearly in r. Let $\lambda_e(r)$ denote this length. We then have that

$$\begin{aligned}\psi_p(r) &= \frac{\Xi(\mathcal{H}_p^r)}{2r} = \frac{\sum_{e \in A} \|e\|}{2r} + \frac{\sum_{e \in B} \lambda_e(r)}{2r} = a\frac{1}{r} + \frac{\hat{b}r + b}{r} \\ &= (a+b)\frac{1}{r} + \hat{b},\end{aligned}$$

where a, b, and $\hat{b}$ are constants. The lemma follows. ☐

Let $\mathbb{V}(\mathcal{H})$ denote the set of sides of $\mathcal{H}$ containing a vertex, and let $\mathbb{v}(\mathcal{H}) = |\mathbb{V}(\mathcal{H})|$. Similarly, let $\mathbb{E}(\mathcal{H})$ denote the set of corners of $\mathcal{H}$ that lie on a trajectory edge, and let $\mathbb{e}(\mathcal{H}) = |\mathbb{E}(\mathcal{H})|$.

▶ **Lemma 4.13.** *There is a hotspot* $\mathcal{H}^*$ *that maximizes* Ψ *such that* $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) \geq 3$.

Proof. Proof by contradiction. Assume that $\mathcal{H}^*$ is a hotspot that maximizes Ψ, with $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) < 3$, and that there is no hotspot $\mathcal{H}$ with $\Psi(\mathcal{H}) \geq \Psi(\mathcal{H}^*)$ and $\mathbb{v}(\mathcal{H}) + \mathbb{e}(\mathcal{H}) > \mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*)$.

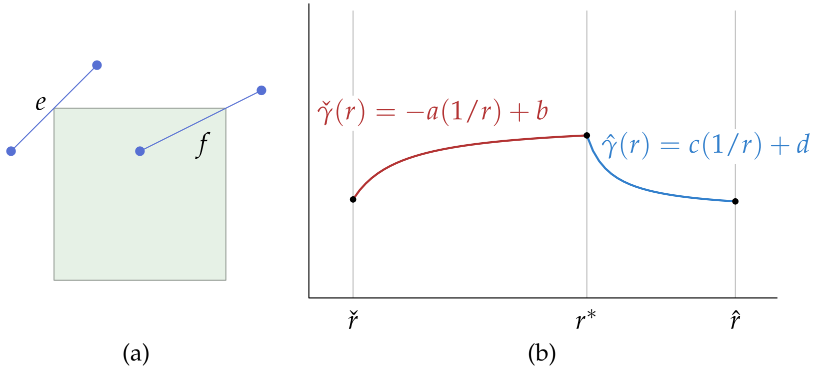
We now show that we can scale or translate $\mathcal{H}^*$ without decreasing Ψ and while keeping $\mathbb{V}(\mathcal{H}^*)$ and $\mathbb{E}(\mathcal{H}^*)$ the same until (i) there is a new vertex on a (new) side of $\mathcal{H}^*$ or, (ii) there is an new edge through a new corner of $\mathcal{H}^*$. This leads to a contradiction, and thus proves the lemma.

Let $\mathcal{H}' = \mathcal{H}^*$, and consider the relative length $\psi(a) = \Psi(\mathcal{H}')$ in $\mathcal{H}'$ as a function of some parameter a. We choose a to be a translation if there are two opposite sides in $\mathbb{V}(\mathcal{H}^*)$, and a scaling otherwise. In both cases we will show that (1) $\mathcal{H}^*$ corresponds to an interior value of a piece π of ψ, (2) one of the endpoints a' of π has $\psi(a') \geq \Psi(\mathcal{H}^*)$, and (3) the break points of ψ correspond to hotspots $\mathcal{H}'$ such that (i) there is a vertex on a side of $\mathcal{H}'$ or, (ii) there is an edge through a corner of $\mathcal{H}'$. It follows that the hotspot $\mathcal{H}'$ corresponding to a' has $\Psi(\mathcal{H}') \geq \Psi(\mathcal{H}^*)$ and $\mathbb{v}(\mathcal{H}') + \mathbb{e}(\mathcal{H}') > \mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*)$, as desired.

Consider the case in which $\mathbb{V}(\mathcal{H}^*)$ contains two opposite sides s_1 and s_2. Assume without loss of generality that s_1 and s_2 are horizontal. We now choose a to be the x-coordinate of the center of $\mathcal{H}'$. We leave the radius of $\mathcal{H}'$ fixed, so ψ is a piecewise linear function in a. This proves (2). The break points of ψ correspond to hotspots such that a vertex of $\mathcal{T}$ lies on a vertical side of $\mathcal{H}'$ or a trajectory edge intersects a corner of $\mathcal{H}'$. This proves (3). Item (1) follows since $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) < 3$.

Consider the case in which $\mathbb{V}(\mathcal{H}^*)$ does not contain two opposite sides. Since $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) < 3$, there is a point $q \in \mathcal{H}^*$ such that if we scale $\mathcal{H}' = \mathcal{H}^*$ by a small amount with q as origin, the vertices on $\partial\mathcal{H}'$ stay on the same side as in $\mathcal{H}^*$, and the edges through corners of $\mathcal{H}'$ go through the same corners in $\mathcal{H}^*$. Let a be the radius of $\mathcal{H}'$ during this scaling. Thus, $\psi(a) = \psi_q(a)$. Items (2) and (3) now directly follow from Lemma 4.12. Item (1) again holds since $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) < 3$. ☐

When a corner c of $\mathcal{H}$ lies in the interior of an edge $e = \overline{uv}$, that is, $c \neq u \neq v$, and $e \cap \mathcal{H} = c$, e *touches* $\mathcal{H}$, see Figure 4.5(a). Let $\mathbb{E}_{nt}(\mathcal{H})$ denote the set of corners of $\mathcal{H}$ that lie on trajectory edges that do not touch $\mathcal{H}$. That is, for each corner c in $\mathbb{E}_{nt}(\mathcal{H})$, the edge through c does not touch $\mathcal{H}$. Let $\mathbb{e}_{nt}(\mathcal{H}) = |\mathbb{E}_{nt}(\mathcal{H})|$.



e
f
$\check{\gamma}(r) = -a(1/r) + b$
$\hat{\gamma}(r) = c(1/r) + d$
$\check{r}$
r^*
$\hat{r}$

(a) (b)

Figure 4.5: (a) Edge e touches the hotspot, edge f does not. (b) The pieces $\check{\gamma}$ and $\hat{\gamma}$ of Γ incident to a maximum r^*.

Lemma 4.14. *There is a hotspot $\mathcal{H}^*$ that maximizes Ψ such that $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}_{nt}(\mathcal{H}^*) \geq 3$.*

Proof. It follows from Lemma 4.13 that there is a hotspot $\mathcal{H}^*$ maximizing Ψ, with $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}(\mathcal{H}^*) \geq 3$. We now show by contradiction that $\mathbb{v}(\mathcal{H}^*) + \mathbb{e}_{nt}(\mathcal{H}^*) \geq 3$.

Let r^* be the radius of $\mathcal{H}^*$, and let e be an edge that touches $\mathcal{H}^*$ in corner c. Hence, $c \in \mathbb{E}_{nt}(\mathcal{H}^*)$. If no such edge, and thus no such corner, exists then we immediately arrive at a contradiction. If e does exist, we will show that this contradicts the fact that $\mathcal{H}^*$ maximizes Ψ.

There is a point $p \in \mathcal{H}^*$ such that we can scale $\mathcal{H}^*$ with p as origin while maintaining a total of two objects $o_1 \neq c$ and $o_2 \neq c$ in $\mathbb{V}(\mathcal{H}^*)$ and $\mathbb{E}(\mathcal{H}^*)$. Now consider the piecewise hyperbolic function ψ_p (Lemma 4.12). Since $\mathcal{H}^*$ is optimal, the value r^* is a break point of ψ_p. Let $[\check{r}, r^*]$ and $[r^*, \hat{r}]$ be the pieces incident to r^*, and let $\check{\psi}$ and $\hat{\psi}$ denote the function ψ_p restricted to their corresponding pieces. See Figure 4.5(b).

Since $\mathcal{H}^*$ maximizes Ψ we have $\psi(r^*) \geq \psi(\check{r}), \psi(\hat{r})$. It follows that $\check{\psi}(r) = -a(1/r) + b$, and $\hat{\psi}(r) = c(1/r) + d$, for some $a, c \geq 0$ and $b, d \in \mathbb{R}$. We now consider the length of edge e in the hotspot as a function of r on the interval $[r^*, \hat{r}]$. Since this length $\lambda_e(r)$ is non-negative, we have $\lambda_e(r) = gr - h$, for some $g, h \geq 0$. This gives us

$$\hat{\psi}(r) = \check{\psi}(r) + \frac{\lambda_e(r)}{2r} = \check{\psi}(r) - \frac{h}{2r} + \frac{g}{2} = (-a - \frac{h}{2})\frac{1}{r} + b + \frac{g}{2}.$$

Hence $c = (-a - h/2) < 0$. Contradiction. ☐

An algorithm to maximize Ψ. By Lemma 4.14 there are three objects, that is, vertices or edges, bounding an optimal hotspot. Hence, there is a hotspot maximizing Ψ such that

- $\partial\mathcal{H}^*$ contains three vertices on three different sides,
- $\partial\mathcal{H}^*$ contains two vertices on different sides and one edge intersects a corner of $\mathcal{H}^*$,
- a vertex lies in a corner of $\mathcal{H}^*$, and there is either a second vertex on $\partial\mathcal{H}^*$ or an edge going through a different corner of $\mathcal{H}^*$,
- two edges intersect in a corner of $\mathcal{H}^*$, and there is either a vertex on $\partial\mathcal{H}^*$ or an edge going through a different corner of $\mathcal{H}^*$,
- $\partial\mathcal{H}^*$ contains one vertex and two edges intersect distinct corners of $\mathcal{H}^*$, or
- three edges intersect distinct corners of $\mathcal{H}^*$.

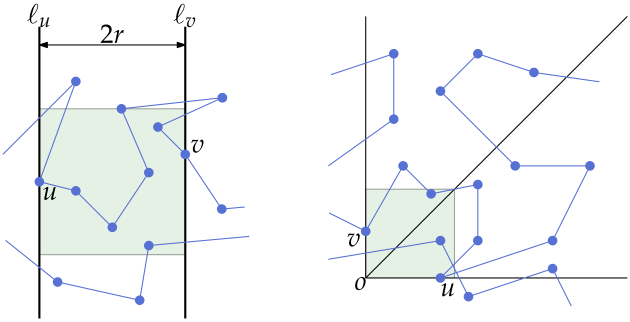
In all cases the edges do not touch $\mathcal{H}^*$.

We now compute a hotspot maximizing Ψ in each of these cases, and then simply pick a best one. In each case, our global approach is to fix two of the three objects. This fixes two of the three degrees of freedom. We then express Ξ as a function of the remaining degree of freedom a. This function Ξ is piecewise linear in a, and will have break points (events) when a third object bounds the hotspot $\mathcal{H}$. Thus, we can find an optimal solution by evaluating Ξ and Ψ at each of the break points.

How we find all break points differs per case, but we will show that in each case we can find the $O(n)$ break points in linear time. Computing and maintaining Ξ also takes linear time in total, since each update can be handled by adding or subtracting a simple linear function that describes the change at that break point. There are $O(n^2)$ pairs of objects that we can fix, so we can handle each case in $O(n^3)$ time in total.

We use the following simple data structures throughout the different cases. We construct two lists $\mathcal{L}_x$ and $\mathcal{L}_y$ of all vertices, $\mathcal{L}_x$ sorted on increasing x-coordinate, and $\mathcal{L}_y$ sorted on increasing y-coordinate. Furthermore, we explicitly build the arrangement $\mathcal{A}$ on the supporting lines of the edges of $\mathcal{T}$. With this arrangement we can now answer the following queries in $O(n)$ time: given a query (half-)line, or *ray*, ℓ find all trajectory edges intersected by ℓ in the order in which they are intersected. We do this by simply walking along ℓ in the arrangement $\mathcal{A}$. Since the zone of ℓ in $\mathcal{A}$ has linear complexity, such a query takes $O(n)$ time.

Three vertices. There is an optimal hotspot such that three sides contain a vertex. Two of these sides must be parallel. Assume without loss of generality that these two sides are contained in the vertical lines ℓ_u and ℓ_v , and that these lines are at distance $2r$ from each other (see Figure 4.6(a)). These two vertical lines bound a vertical slab, we now place a square hotspot $\mathcal{H}$ with radius r at



(a)

(b)

Figure 4.6: (a) The three vertices case. (b) The two vertices, one edge case.

the bottom of this slab, and shift it upwards. Let a be the y-coordinate of the top of $\mathcal{H}$, and consider Ξ as a function of a. Each time a side of $\mathcal{H}$ hits a vertex, or a corner of $\mathcal{H}$ hits a trajectory edge, we get a break point.

Two vertices, one edge. Let u and v be the vertices on $\partial\mathcal{H}^*$. When u and v lie on opposite sides of $\mathcal{H}^*$ we can use the same approach as in the previous case. When u and v lie on neighboring sides we use the following approach. Assume without loss of generality that u lies on the bottom side of $\mathcal{H}^*$ and v on the left side of $\mathcal{H}^*$. This means that the bottom left corner o of $\mathcal{H}^*$ is uniquely defined by u and v.

We start with an arbitrarily small empty hotspot $\mathcal{H}$ with its bottom left corner at o. We now scale $\mathcal{H}$ with o as origin. Let a denote the scaling parameter, and consider Ξ as a function of a. The function Ξ combinatorially changes when any side of $\mathcal{H}$ hits a vertex, or any corner of $\mathcal{H}$ hits a trajectory edge. We find these times by querying $\mathcal{A}$ with a horizontal ray, a vertical ray, and a diagonal ray starting at o, and merging their results with $\mathcal{L}_x$ and $\mathcal{L}_y$ (see Figure 4.6(b)).

One corner vertex. Let v be the vertex of $\mathcal{T}$ on a corner of $\mathcal{H}^*$. We define $o = v$, and then handle this case analogous to the previous case.

Two edges through a single corner. Let e and f be the two edges that intersect in point o. Thus, o is the corner of $\mathcal{H}^*$. We then again handle this case analogous to the two vertices, one edge case.

One vertex, two edges. Let v be the vertex on $\partial\mathcal{H}^*$, and let e be the edge through a corner of $\mathcal{H}^*$. Assume without loss of generality that v lies on the bottom side of $\mathcal{H}^*$. We distinguish two subcases: e intersects a bottom corner of $\mathcal{H}^*$, or e intersects a top corner of $\mathcal{H}^*$.

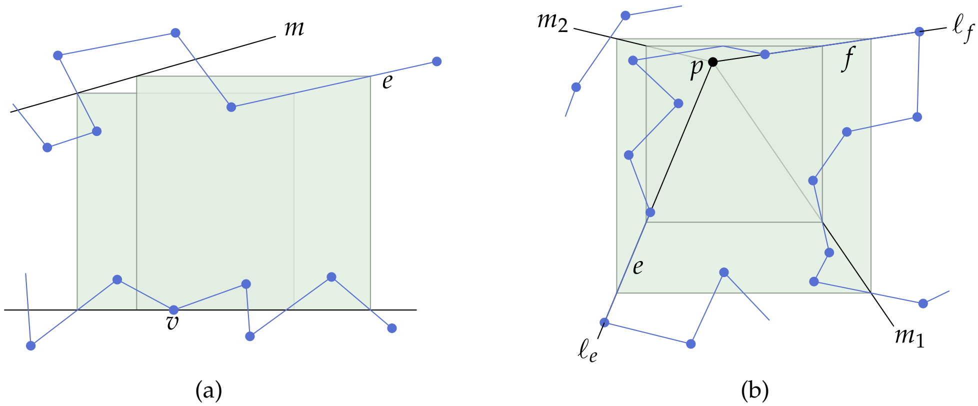


Figure 4.7: The one vertex, two edges case (a), and the three edges case (b).

In the first case, the horizontal line through v intersects e in a point o. We again consider scaling $\mathcal{H}$ with origin o, so this case is handled analogously to the case where there were two vertices on $\partial\mathcal{H}^*$.

In the second case, e intersects $\mathcal{H}$ in a top corner c_1. Let c_2 be the other top corner. All hotspots that have corner c_1 on e, and v on the bottom side, have their other upper corner, c_2, on a line m (see Figure 4.7(a)). We consider these hotspots and the function Ξ by increasing size a. We find the break points as follows. To find all edges that could intersect $\mathcal{H}$ in corner c_2, we query $\mathcal{A}$ with m, oriented such that the hotspots $\mathcal{H}$ get larger along m. We find the intersections with other corners, ordered by increasing size a, by querying $\mathcal{A}$ with two horizontal half-lines starting at v. We then merge these three lists with the sorted lists of vertices to get a list of all break points.

Three edges. Let e,f, and g be the edges through corners of $\mathcal{H}^*$, of which e and f are through opposite corners c_e and c_f. Let ℓ_e and ℓ_f be the lines containing e and f, respectively (see Figure 4.7(b)). Consider all hotspots that have e through corner c_e and f through c_f. The remaining two corners of these hotspots lie on half-lines m_1 and m_2, starting at the intersection point p of ℓ_e and ℓ_f.

We now consider Ξ as a function of the size a of the hotspots that have e through corner c_e and f through corner c_f. To compute the break points of Ξ, we query $\mathcal{A}$ with m_1, m_2, ℓ_e and ℓ_f. We merge these lists with the sorted lists of vertices to get all break points, ordered on increasing size a.

We now conclude:

▶ **Theorem 4.15.** *We can find a hotspot $\mathcal{H}^*$ that maximizes Ψ in $O(n^3)$ time.*

4.4.2 Contiguous Length

We now consider maximizing the function $\Gamma(\mathcal{H}) = \Phi(\mathcal{H})/w$, where $w = 2r$. Since Φ is piecewise linear in c and r, function Γ is a piecewise function that is linear in c and hyperbolic in r. We again use $\mathcal{T}^* = \mathcal{T}[p^*, q^*]$ to denote a maximal length sub-trajectory contained in an optimal hotspot $\mathcal{H}*$.

As before, we observe that unless p^* is the starting point of $\mathcal{T}$, it lies on the boundary of $\mathcal{H}^*$. We again handle the case $p^* = \mathcal{T}(0)$ separately. This is easy to do in linear time by computing a function W expressing the minimum width of a hotspot containing $\mathcal{T}[0, q^*]$ as a function of the ending time q^*_t. We can then easily compute (the maximum of) Γ using W. In the remainder of this section we will therefore assume that p^* lies on the boundary of $\mathcal{H}^*$.

▶ **Lemma 4.16.** *There is a hotspot $\mathcal{H}^*$ maximizing Γ, with sides $s_0, .., s_3$ in clockwise order such that:*

- *There is a vertex $v \in \mathcal{T}^*$ on a side $s_{(i+1) \bmod 4}$, and*
- *$\mathcal{T}[p^*, v]$ or $\mathcal{T}[v, q^*]$ intersects side s_i.*

Proof. Let w^* denote the width of a hotspot maximizing Γ. By Lemma 4.7, applied to hotspots of width w^*, there is a hotspot $\mathcal{H}^*$ (of width w^*) that maximizes Φ and has a vertex $v \in \mathcal{T}^*$ on its boundary. It follows that $\mathcal{H}^*$ also maximizes Γ.

Let the side on which v lies be $s_{(i+1) \bmod 4}$, and assume without loss of generality that $s_{(i+1) \bmod 4}$ is the top side of $\mathcal{H}^*$. Clearly, we can now just shift $\mathcal{H}^*$ to the right, while keeping $\mathcal{T}^*$ inside it, until $\mathcal{T}^*$ intersects the left side s_i. Since $v \in \mathcal{T}^*$ it thus also holds that $\mathcal{T}[p^*, v]$ or $\mathcal{T}[v, q^*]$ (or both) intersect side s_i. ☐

▶ **Corollary 4.17.** *There is a vertex $v \in \mathcal{T}$, and a hotspot $\mathcal{H}^*$ that maximizes Γ such that:*

- *v lies on side $s_{(i+1) \bmod 4}$ of $\mathcal{H}^*$,*
- *$p^*_x \in \{v_x - w^*, v_x, v_x + w^*\}$ or $p^*_y \in \{v_y - w^*, v_y, v_y + w^*\}$, and*
- *$\mathcal{T}[p^*, v]$ or $\mathcal{T}[v, q^*]$ intersects side s_i.*

From Corollary 4.17 it follows that we can use a similar approach as in Section 4.3.1: we fix vertex v, a side $s_{(i+1) \bmod 4}$ of $\mathcal{H}^*$, which part of $\mathcal{T}$ intersects s_i (either the part before or after v), and one of the six constraints on p^*, and we compute Γ and its maximum, assuming that there is an optimal hotspot with that configuration. To compute and find $\Gamma_{\mathcal{C}} = \Gamma$ for a given configuration $\mathcal{C}$, we express the maximal contiguous length as a function of w. This also gives us the function $\Gamma_{\mathcal{C}}(w) = \Phi_{\mathcal{C}}(w/2)/w$. Since $\Phi_{\mathcal{C}}$ is piecewise linear, $\Gamma_{\mathcal{C}}$ is piecewise hyperbolic. We can thus find the maximum of $\Gamma_{\mathcal{C}}$ by inspecting each

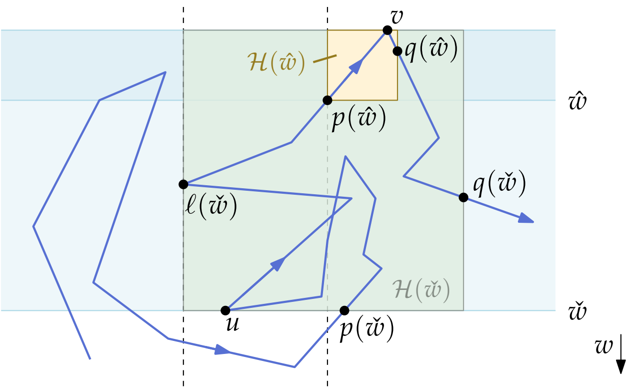


Figure 4.8: Vertex v together with a width w define the functions $p(w)$, $\ell(w)$, and $q(w)$. In turn, these functions can be used to define a hotspot $\mathcal{H}(w)$ and the length therein. The figure illustrates these functions for two parameter values $\hat{w}$ and $\check{w}$.

piece. We will show that we can do this in $O(n)$ time. Since there are $O(n)$ configurations, this leads to an $O(n^2)$ time algorithm.

Consider a configuration $\boldsymbol{C}$ in which v lies on the top side of an optimal hotspot $\mathcal{H}^*$, where p^* lies on a horizontal side of $\mathcal{H}^*$ (hence $p^*_y \in \{v_y - w^*, v_y\}$), and $\mathcal{T}[p^*, v]$ intersects the left side of $\mathcal{H}^*$. All other cases can be handled analogously. Let $S(w) = [v_y - w, v_y]$ denote the horizontal slab of width w and with v on it's top boundary, let $p(w)$ be the earliest point on $\mathcal{T}$, that is, the point with the earliest associated time, such that $\mathcal{T}[p(w), v]$ is completely contained in $S(w)$, and let $\ell(w)$ be the leftmost point in $\mathcal{T}[p(w), v]$. See Figure 4.8.

We now observe that when vertex v together with a width w uniquely define a hotspot $\mathcal{H}(w)$: $\mathcal{H}(w)$ has width w, $\ell(w)$ on its left side, and v on its top side. By construction, $\mathcal{T}$ enters $\mathcal{H}(w)$ through a horizontal side, namely in point $p(w)$. It follows that $\mathcal{H}^* = \mathcal{H}(w^*)$, for the optimal width w^*. Let $q(w)$ be the latest point on $\mathcal{T}$ such that $\mathcal{T}[p(w), q(w)] \subset \mathcal{H}(w)$. It follows that $\Phi_{\boldsymbol{C}}(w) = \|\mathcal{T}[p(w), q(w)]\|$, and $\Gamma_{\boldsymbol{C}}(w) = \Phi_{\boldsymbol{C}}(w)/w$. Note that if $\ell(w)_x + w < v_x$ then $\mathcal{H}(w)$ does not exist, so $\mathcal{H}$, is a partial function. Therefore q, $\Phi_{\boldsymbol{C}}$, and $\Gamma_{\boldsymbol{C}}$ are also partial functions.

It is easy to see that p is piecewise linear in w, and that its break points correspond to vertices of $\mathcal{T}$. Hence p consists of at most $O(n)$ pieces. It then follows that ℓ, q, and $\Phi_{\boldsymbol{C}}$ are all also piecewise linear, and consist of $O(n)$ pieces. This, in turn, implies that $\Gamma_{\boldsymbol{C}}$ is piecewise hyperbolic and consists of $O(n)$ pieces. Next, we describe how construct $\Gamma_{\boldsymbol{C}}$ in linear time.

Computing $\Gamma_{\boldsymbol{C}}$ for a given configuration. We first construct p. We start with $w = 0$ and increase w while maintaining p. Increasing w corresponds to walking along the trajectory, starting at $w(0) = v$, and heading towards the starting point of $\mathcal{T}$. When we encounter a new edge e oriented upwards (and thus downwards with respect to our walk) $p(w)$ is a simple linear function depending on e. When we encounter an vertex of an edge e oriented downwards (like at vertex u in Figure 4.8), $p(w)$ is either u, or the last point before u where $p(w)_y = u_y$. Hence, we can simply walk along $\mathcal{T}$ until we find it, we reach the global starting point of $\mathcal{T}$, or we reach a vertex with a y-coordinate greater than v_y. In the latter two cases we set $p(w) = u$, and $p(w') = \perp$ (undefined) for any $w' > w$, and we are done. In the former case, we set $p(w)$ to be the point found, and continue increasing w (since we know that this point lies on an upward oriented edge).

The procedure above takes $O(n)$ time. Furthermore, it is easy to extend it to maintain ℓ as well. A very similar approach can then be used to compute q. Since q is also monotone in w, this also takes linear time. When we have the functions p and q, Φ_C, Γ_C, and the maximum of Γ_C can be computed in $O(n)$ time. We use this procedure for all $O(n)$ configurations. Thus:

▶ **Theorem 4.18.** *We can find a hotspot $\mathcal{H}^*$ that maximizes Γ in $O(n^2)$ time.*

4.5 Extensions

In this section we briefly discuss various extensions to our algorithms, in particular, multiple entities and differently shaped hotspots.

Multiple entities. Suppose that instead of one moving entity with one trajectory $\mathcal{T}$, we have many entities, and thus many trajectories $\mathcal{T}_1, .., \mathcal{T}_m$. Can we still find a hotspot $\mathcal{H}$ that maximizes the length (over all trajectories) inside it, or minimizes the size required to get at least a certain length? Our algorithms simply treat trajectory $\mathcal{T}$ as a set of line segments. So they are immediately applicable to multiple trajectories as well. Note that for the contiguous-length versions we now maximize over all input trajectories as well.

Convex polygonal hotspots of a given shape. The algorithms that use the total length (the ones from Section 4.2), are still applicable when the hotspot is a convex polygon of a given shape. If the polygon has k vertices, each trajectory edge produces $O(k)$ line segments in subdivision $\mathcal{S}$. Thus, $\mathcal{S}$ may have a total complexity of $O(k^2n^2)$. We then solve the fixed radius version (for an appropriate definition of radius) in $O(k^2n^2)$ time. Similarly, the fixed length version takes $O(k^2n^2 \log^2(kn))$ time.

If we consider the relative total length inside $\mathcal{H}$, Lemmas 4.13 and 4.14 still hold, even if $\mathcal{H}$ is a convex polygon of complexity k of a given shape. This means that there are still three objects "bounding" $\mathcal{H}^*$; one for each parameter specifying the position of $\mathcal{H}^*$. So, the global approach of our algorithm is still applicable. Which, and how many, ray shooting queries we have to perform to obtain the break points depends on the shape of the hotspot.

For the algorithms that use the contiguous length (time) it is not immediately clear how to extend them to work for arbitrarily, but fixed, shaped polygons.

Other types of hotspots. When the hotspot has curved boundaries, we can no longer maximize our functions Ξ and Φ (and as a result Ψ and Γ), as is explained in Section 4.1. Hence, our algorithms do not easily extend to these cases. When the hotspot $\mathcal{H}$ is not convex, the intersection of a single edge with $\mathcal{H}$ may consist of several line segments. This will increase the time required to evaluate the functions. Furthermore, this may lead to a large increase of complexity in the parameter space. When the shape of the hotspot is not predefined it is not clear how define the problem as a maximization problem. Hence, in this case we cannot directly apply our algorithms either.

Computing c-packedness. We note that our measure Ψ of relative length is closely related to the "c-packedness" of a curve. A curve $\boldsymbol{C}$ in $\mathbb{R}^2$ is c-packed if and only if for any point $p \in \mathbb{R}^2$ and any radius r the total length in the disk of radius r centered at p is at most cr [43]. If we use a square instead of a disk, then $\boldsymbol{C}$ is c-packed if and only if $\Psi(r)/2 \leq c$. Hence, for this modified notion of c-packed curves, our algorithm from Section 4.4 can compute the smallest c for which a curve is c-packed in $O(n^3)$ time. For the original definition of c-packed we need a round hotspot, and thus run into the same issues as described above.

4.6 Concluding Remarks

Hotspots are small regions where a moving entity spends a significant amount of time. We presented algorithms to locate optimal hotspots from trajectory data based on path length rather than time, but versions based on time require essentially the same algorithms and the same efficiency is obtained. Five variations of the problem were considered. When all visits to the hotspot count our algorithms take roughly quadratic time: $O(n^2)$ time if we want to maximize the time in a fixed-size square hotspot, and $O(n^2 \log^2 n)$ time if we want to minimize the size of the hotspot for a fixed time the entity spends inside. Maximizing the relative time inside, compared to the size of the hotspot, takes $O(n^3)$ time. If we are interested only in the longest contiguous visit, we can improve on these running times. Solving the fixed length and fixed radius versions takes only $O(n \log n)$ time, and computing a hotspot that maximizes the relative contiguous length takes $O(n^2)$ time.

There are various heuristic extensions possible to let the algorithm with cubic runtime be much faster in practice. For all algorithms we can apply trajectory simplification to improve the efficiency if needed, because our methods can handle irregularly sampled data without problems (unlike point-based methods).

Our algorithms directly extend to multiple entities. However, when multiple entities are considered several other variations of the problem exist. For example, find a smallest hotspot $\mathcal{H}$ such that all entities spend at least L time in $\mathcal{H}$. We can again consider the total time, the longest contiguous time, and we can even vary whether or not the entities need to be present at the same time(s). Note that the contiguous time version in which all entities need to be present at the same time is very related to the meet pattern as defined by Gudmundsson and van Kreveld [60]. We can also consider only the longest visit to the hotspot for each entity, and try to maximize the sum of those durations over all entities. All these variations are interesting options for future work. Additionally, we can consider computing hotspots for entities moving in $\mathbb{R}^d$, with $d \geq 3$, or finding a set of hotspots.

Part III

Analysis Tasks for Multiple Trajectories

Chapter 5

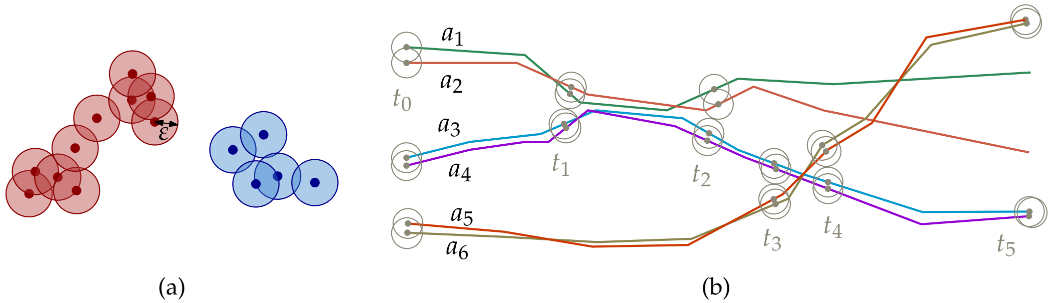
The Trajectory Grouping Structure

In this chapter we present a way to characterize and compute *groups* in a set of trajectories of moving entities. A group is a sufficiently large set of entities that travel together for a sufficiently long time (we give a more formal definition later). Groups may start, end, split and merge with other groups. Apart from the question what the current groups are, we also want to know which splits and merges led to the current groups, when they happened, and which groups they involved. We wish to capture this group change information in a model that we call the *trajectory grouping structure*. We define the trajectory grouping structure using the *Reeb graph*, a concept from topology [47].

The description above suggests that three parameters are needed to define groups: (i) a spatial parameter for the distance between entities; (ii) a temporal parameter for the duration of a group; (iii) a count for the number of entities in a group. We will design our grouping structure definition to incorporate these parameters so that we can study grouping at different scales. We also discuss robustness of the grouping structure in the following sense. If an entity a leaves a group G and almost immediately returns, we would like to ignore the small time during which a and G were separate, and just consider $G \cup \{a\}$ as one group. This requires one additional parameter that captures how short any interruption in a group may last to be ignored.

A definition for a group. Let $\mathcal{E}$ be a set of entities for which we have the trajectories. The *ε-disc* of an entity a (at time t) is a disc of radius ε centered at a at time t. Two entities are *directly connected* at time t if their ε-discs overlap. Two entities a and b are *ε-connected* at time t if there is a sequence $a = a_0, .., a_k = b$ of entities such that for all i, a_i and a_{i+1} are directly connected.

A subset $S \subseteq \mathcal{E}$ of entities is ε-connected at time t if all entities in S are pairwise ε-connected at time t. This means that the union of the ε-discs of entities in S forms a single connected region. The set S forms a *component* at time t if and only if S is ε-connected, and S is maximal with respect to this property. See Figure 5.1(a). The set of components $\boldsymbol{C}(t)$ at time t forms a partition of the entities in $\mathcal{E}$ at time t.



(a) (b)

Figure 5.1: (a) The entities and their ε-discs at a particular time t. The entities form two maximal ε-connected sets, or components, at time t; a red component and a blue component. (b) For $m = 2$ and $t_2 - t_1 \geq \delta > t_4 - t_3$ there are four maximal groups: $\{a_1, a_2\}$, $\{a_3, a_4\}$, $\{a_5, a_6\}$, and $\{a_1, .., a_4\}$.

Let the spatial parameter of a group be ε, the temporal parameter δ, and the size parameter m. A set G of k entities forms a *group* during time interval I if and only if the following three conditions hold: (*i*) G contains at least m entities, so $k \geq m$, (*ii*) the interval I has length at least δ, and (*iii*) at all times $t \in I$, there is a component $C \in \boldsymbol{C}(t)$ such that $G \subseteq C$.

We denote the interval $I = [t_s, t_e]$ of group G with I_G. Group H *covers* group G if $G \subseteq H$ and $I_G \subseteq I_H$. If there are no groups that cover G, we say G is *maximal* (on I_G). In Figure 5.1(b), groups $\{a_1, a_2\}$, $\tilde{G} = \{a_3, a_4\}$, $\hat{G} = \{a_5, a_6\}$, and $G = \{a_1, .., a_4\}$ are maximal: $\tilde{G}$ and $\hat{G}$ on $[t_0, t_5]$, G on $[t_1, t_2]$. Group $\{a_1, a_3\}$ is covered by G and hence not maximal.

Note that entities can be in multiple maximal groups at the same time. For example, entities $\{b_1, b_2, b_3\}$ can travel together for a while, then b_4, b_5 may become ε-connected, and shortly thereafter b_1, b_4, b_5 separate and travel together for a while. Then b_1 may be in two otherwise disjoint maximal groups for a short time. An entity can also be in two maximal groups where one is a subset of the other. In that case the group with fewer entities must last longer. That an entity is in more groups simultaneously may seem counterintuitive at first, but it is necessary to capture all grouping information. We will show that the total number of maximal groups is $O(\tau n^3)$, where n is the number of entities in $\mathscr{E}$ and τ is the number of edges of each input trajectory. This bound is tight in the worst case.

Our maximal group definition uses three parameters, which all allow a more global view of the grouping structure. In particular, we observe that there is *monotonicity* in the group size and the duration: If G is a group during interval I, and we decrease the minimum required group size m or decrease the minimum required duration δ, then G is still a group on time interval I. Also, if G is a maximal group on I, then it is also a maximal group for a smaller m or smaller δ. For the spatial parameter ε we observe monotonicity in a slightly different manner: if G is a group for a given ε, then for a larger value of ε there exists a group $G' \supseteq G$. The monotonicity property is important when we want to have a more detailed view of the data: we do not lose maximal groups in a more detailed view. The group may, however, be extended in size and/or duration.

We capture the grouping structure using a Reeb graph of the ε-connected components together with the set of all maximal groups. Parts of the Reeb graph that do not support a maximal group can be omitted. The grouping structure can help us in answering various questions. For example:

- What is the largest/longest maximal group at time t?
- How many entities are currently (not) in any maximal group?
- What is the first maximal group that starts/ends after time t?
- What is the total time that an entity was part of any maximal group?
- Which entity has shared maximal groups with the most other entities?

Furthermore, the grouping structure can be used to partition the trajectories in independent data sets, to visualize grouping aspects of the trajectories, and to compare grouping across different data sets.

Results and organization. We discuss how to represent the grouping structure using the Reeb graph in Section 5.1. We prove that for n entities, each traveling along a trajectory of τ edges, the Reeb graph has size at most $O(\tau n^2)$, and that there are at most $O(\tau n^3)$ maximal groups. Both of these bounds are tight in the worst case. We present algorithms to compute the Reeb graph and the trajectory grouping structure and all maximal groups in Section 5.2. Computing the Reeb graph takes $O(\tau n^2 \log n)$ time, and computing all maximal groups takes $O(\tau n^3 + N)$ time, where N is the total output size. The maximal group definition given above is not yet robust, in Section 5.3 we incorporate robustness and extend our algorithms to this case. In Section 5.4 we evaluate our methods on synthetic and real-world data.

5.1 Representing the Grouping Structure

Let $\mathcal{E}$ be a set of n entities, where each entity travels along a path of τ edges. To compute the grouping structure we consider a manifold $\mathcal{M}$ in $\mathbb{R}^3$, where the z-axis corresponds to time. The manifold $\mathcal{M}$ is the union of n "tubes" (see Figure 5.2(a)). Each tube consists of τ skewed cylinders with horizontal radius ε that we obtain by tracing the ε-disc of an entity x over its trajectory.

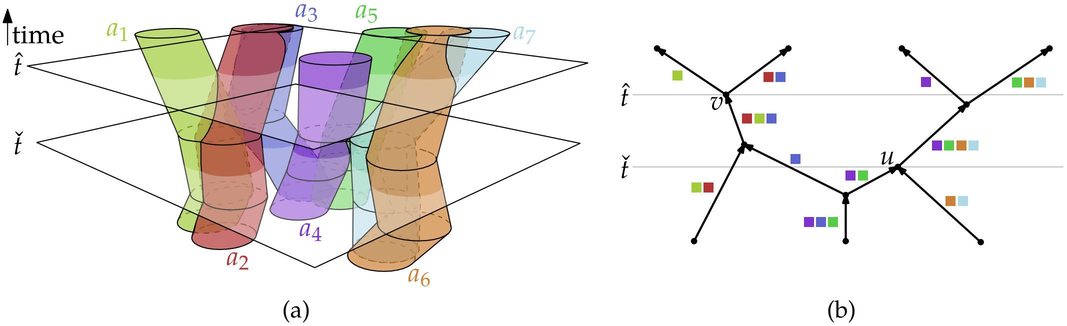


Figure 5.2: (a) The manifold $\mathcal{M}$ for the entities $\mathcal{E} = \{a_1, .., a_7\}$. At time $\check{t}$ the component $\{a_4, a_5\}$ merges with $\{a_6, a_7\}$, and at time $\hat{t}$ the group $\{a_1, a_2, a_3\}$ splits. (b) The Reeb graph $\mathcal{R}$ corresponding to $\mathcal{M}$. The colored squares indicate which entities are in the component associated with each edge of $\mathcal{R}$. The merge of $\{a_4, a_5\}$ and $\{a_6, a_7\}$ yields a merge vertex u at time $u_t = \check{t}$, and the split of $\{a_1, a_2, a_3\}$ yields a split vertex v at time $t_v = \hat{t}$.

Let H_t denote the horizontal plane at height t, then the set $\mathcal{M} \cap H_t$ is the *level set* of t. The connected components in the level set of t correspond to the components (maximal sets of ε-connected entities) at time t. To simplify the presentation we will assume that all trajectories have their known positions at the same times $t_0, .., t_\tau$ and that no three entities become ε-(dis)connected at the same time, but our theory can easily be extended to remove these assumptions.

5.1.1 The Reeb Graph

We start out with a possibly disconnected solid that is the union of a collection of tube-like regions: a 3-manifold with boundary. Note that this manifold is not explicitly defined. We are interested in horizontal cross-sections, and the evolution of the connected components of these cross-sections defines the Reeb graph. Note that this is different from the usual Reeb graph that is obtained from the 2-manifold that is the boundary of our 3-manifold, using the level sets of the height function (the function whose level sets we follow is the height function above a horizontal plane below the manifold), see [47] for a background on these topics.

To describe how the components change over time, we consider the Reeb graph $\mathcal{R}$ of $\mathcal{M}$ (Figure 5.2(b)). The Reeb graph has a vertex v at every time t_v where the components change. The vertex times are usually not at any of the given times $t_0, .., t_\tau$, but in between two consecutive time steps. The vertices of the Reeb graph can be classified in four groups. There is a *start vertex* for every component at t_0 and an *end vertex* at t_τ. A start vertex has in-degree

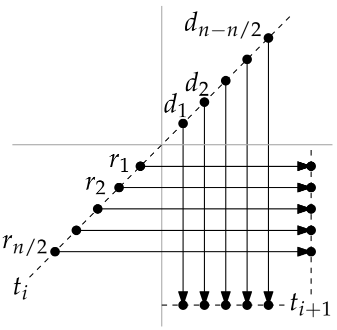


Figure 5.3: Every pair of entities r_j and d_ℓ are at the same point at time $t_i + j + \ell$. This yields $\Omega(n^2)$ vertices in the interval $[t_i, t_{i+1}]$.

zero and out-degree one, and an end vertex has in-degree one and out-degree zero. The remaining vertices are either *merge vertices* or *split vertices*. Since we assume that no three entities become ε-(dis)connected at exactly the same time there are no simultaneous splits and merges. This means merge vertices have in-degree two and out-degree one, and split vertices have in-degree one and out-degree two. A directed edge $e = (u, v)$ connecting vertices u and v, with $t_u < t_v$, corresponds to a set C_e of entities that form a component at any time $t \in I_e = [t_u, t_v]$. The Reeb graph is this directed graph. Note that the Reeb graph depends on the spatial parameter ε, but not on the other two parameters of maximal groups.

▶ **Lemma 5.1.** *The Reeb graph $\mathcal{R}$ for a set $\mathcal{E}$ of n entities, each of which travels along a trajectory of τ edges, can have $\Omega(\tau n^2)$ vertices and $\Omega(\tau n^2)$ edges.*

Proof. Assume without loss of generality that n is even. We construct n trajectory edges on which the entities travel in between two consecutive time stamps, say t_i and t_{i+1}, such that the Reeb graph for $\varepsilon = 0$ has $\Omega(n^2)$ vertices v with $t_v \in [t_i, t_{i+1}]$. We use this construction in between all times t_{2i} and t_{2i+1}, and move the entities back to their starting position in between t_{2i+1} and t_{2i+2}. Therefore, the total number of vertices is $\Omega(\tau n^2)$. Since each vertex has degree one or three it follows that the number of edges is also $\Omega(\tau n^2)$.

Let $\mathcal{E} = R \cup D$, with $R = r_1, .., r_{n/2}$ and $D = d_1, .., d_{n-n/2}$. At the start (time t_i) all entities start at the line $y = x$. In particular, we place r_j on $(-j, -j)$ and d_ℓ on (ℓ, ℓ). All entities move with speed one. The entities in R move to the right, and the entities in D move downwards (see Figure 5.3). It follows that each entity r_j and d_ℓ are both at the same point at time $t_i + j + \ell$. Hence, we get a vertex in the Reeb graph. There are $\Omega(n^2)$ such intersections, and thus $\Omega(n^2)$ vertices. The lemma follows. ☐

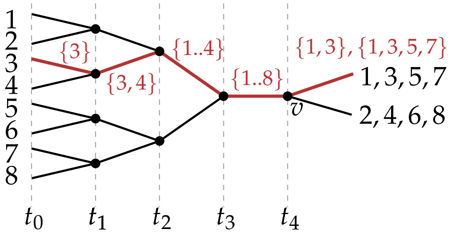


Figure 5.4: The maximal groups containing entity 3 (red). Vertex v creates six new groups, including $\{1,3\}$ and $\{1,3,5,7\}$.

▶ **Theorem 5.2.** *Given a set $\mathcal{E}$ of n entities, in which each entity travels along a trajectory of τ edges, the Reeb graph $\mathcal{R}$ has $O(\tau n^2)$ vertices and edges. This bound is tight in the worst case.*

Proof. Lemma 5.1 gives a simple construction that shows that the Reeb graph may have $\Omega(\tau n^2)$ vertices and edges. For the upper bound, consider a trajectory edge (v_i, v_{i+1}) of (the trajectory of) entity $a \in \mathcal{E}$. During interval $[t_i, t_{i+1}]$, the distance between a and any other entity $b \in \mathcal{E}$ is a convex (hyperbolic) function in t, so b is directly connected to a during at most one interval $I \subseteq [t_i, t_{i+1}]$. This interval yields at most two vertices in $\mathcal{R}$. The trajectory of a consists of τ edges, hence a pair a, b produces $O(\tau)$ vertices in $\mathcal{R}$. This gives a total of $O(\tau n^2)$ vertices. Each vertex has constant degree, so there are $O(\tau n^2)$ edges. ☐

▶ Remark 5.3. For any constant d, the distance between two entities a and b, each moving along a line in $\mathbb{R}^d$, is a convex (hyperbolic) function. Hence, the above result also holds for entities moving in $\mathbb{R}^d$, with $d > 2$.

The trajectory grouping structure. The trajectories of entities are associated with the edges of the Reeb graph in a natural way. Each entity follows a directed path in the Reeb graph from a start vertex to an end vertex. Similarly, (maximal) groups follow a directed path from a start or merge vertex to a split or end vertex. If $m > 0$ or $\delta > 0$, there may be edges in the Reeb graph with which no group is associated. These edges do not contribute to the grouping structure, so we can discard them. The remainder of the Reeb graph we call the *reduced Reeb graph*, which, together with all maximal groups associated with its edges, forms the *trajectory grouping structure*.

5.1.2 Bounding the Number of Maximal Groups

To bound the total number of maximal groups, we study the case where $m = 1$ and $\delta = 0$, because larger values can only reduce the number of maximal groups. It may seem as if each vertex in the Reeb graph simply creates as many maximal groups as it has outgoing edges. However, consider for example Figure 5.4.

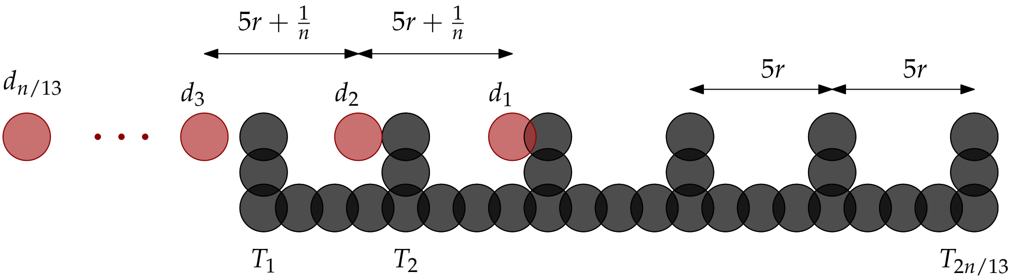


Figure 5.5: An illustration of the lower bound construction at time j_2 in round three. The black discs correspond to the stationary entities in S. The red discs correspond the entities in D.

Split vertex v creates not only the maximal groups $\{1,3,5,7\}$ and $\{2,4,6,8\}$, but also $\{1,3\}$, $\{5,7\}$, $\{2,4\}$, and $\{6,8\}$. These last four groups are all maximal on $[t_2,t]$, for $t > t_4$. Notice that all six newly discovered groups start strictly before t_v, but only at t_v do we realize that these groups are maximal, which is the meaning that should be understood with "creating maximal groups". This example can be extended to arbitrary size. Hence a vertex v may create many new maximal groups, some of which start before t_v. We continue to show that we may obtain $\Omega(\tau n^3)$ maximal groups, and that it cannot get worse than that, that is, the number of maximal groups is at most $O(\tau n^3)$ as well.

▶ **Lemma 5.4.** *For a set $\mathcal{E}$ of n entities, in which each entity travels along a trajectory of τ edges, there can be $\Omega(\tau n^3)$ maximal groups, each of size $\Omega(n)$.*

Proof. Similar to Lemma 5.1 we construct n trajectory edges on which the entities travel in between t_u and t_{u+1}, and repeat this construction in $\Omega(\tau)$ time intervals. Our construction yields $\Omega(n^3)$ maximal groups G with $I_G \subseteq [t_u, t_{u+1}]$, resulting in $\Omega(\tau n^3)$ maximal groups overall as claimed.

For ease of notation we assume (without loss of generality) that n is divisible by thirteen, and we write a to denote both the entity a and the ε-disc of entity a. We partition our set of entities $\mathcal{E}$ into two sets S and D of sizes $\frac{12n}{13}$ and $\frac{n}{13}$, respectively. We first construct S, whose entities are stationary. We place $\frac{8n}{13}$ entities from S on the line $y=0$, with a distance r, $\varepsilon < r < 2\varepsilon$, in between two consecutive entities. On every fifth stationary entity s_i we build a *tower* T_i, that is, we place two more stationary entities vertically above s_i. We place them such that the distance between two consecutive entities is r. See Figure 5.5. Note that S contains $\frac{2n}{13}$ towers, and that all entities in S are ε-connected.

The remaining entities D will move on a horizontal line $y = 3r$. At time t_u, the discs $D = \{d_1, .., d_{n/13}\}$, ordered from right to left, all lie to the left of

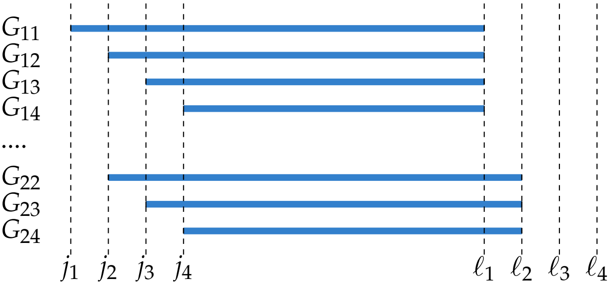


Figure 5.6: The time intervals on which G_{ab} is a maximal group in a given round.

the discs in S. They all move to the right with the same speed. The distance between two consecutive entities is $5r + \frac{1}{n}$.

The sequence of events in interval $[t_u, t_{u+1}]$ can then be partitioned into *rounds*. Round z starts with k merge events $j_1, .., j_k$, followed by a series of k split events $\ell_1, .., \ell_k$. More specifically, at time ℓ_i, entity d_i becomes directly connected with (the topmost entity of) tower T_{1+z-i}, and at time ℓ_i entity d_i stops being directly connected with T_{1+z-i}.

Merge j_i will start a new maximal group G_{1i}, where $G_{ab} = S \cup \bigcup_{h=a}^{b} d_h$. Hence after the k merges, k maximal groups have started. In the subsequent series of split events, the discs $d_1, .., d_k$ stop being directly connected with their corresponding tower. When d_i leaves, the sets of entities $G_{ii}, .., G_{ik}$ end as maximal groups. However, when d_i leaves G_{ih}, it creates $G_{(i+1)h}$ as a new maximal group that started on j_{i+1} (see Figure 5.6). This means ℓ_i creates $k - i$ new maximal groups. Hence, round z creates a total of $\sum_{i=1}^{k}(k-i) = \Omega(k^2)$ maximal groups.

We now show that, for any $m \leq \frac{12n}{13}$ and any δ, this construction yields $\Omega(n^3)$ maximal groups. Since we can choose the speed of the discs in D, we can choose it such that all groups have a minimum duration of at least δ. Now consider the rounds $\frac{n}{13}, .., \frac{2n}{13}$. In each of these $\frac{n}{13}$ rounds we have $k = \frac{n}{13}$. Hence, each round creates $\Omega(n^2)$ new maximal groups. This yields a total of $\Omega(n^3)$ maximal groups. Since each group contains S, its size is at least $\frac{12n}{13}$. ☐

▶ **Theorem 5.5.** *Let $\mathcal{E}$ be a set of n entities, in which each entity travels along a trajectory of τ edges. There are at most $O(\tau n^3)$ maximal groups, and this is tight in the worst case.*

Proof. Lemma 5.4 gives a construction that shows that there may be $\Omega(\tau n^3)$ maximal groups.

We proceed with the upper bound. Every maximal group starts either at a start vertex, or a merge vertex. We will show that the number of maximal

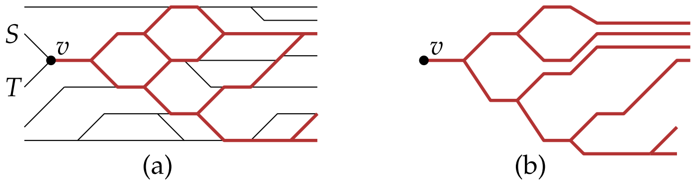


Figure 5.7: DAG $\mathcal{R}'_v$ (red) as a subgraph of $\mathcal{R}$ (black) (a), and the tree $\mathcal{T}_v$ obtained by unfolding $\mathcal{R}'_v$ (b).

groups starting at a start or merge vertex is $O(n)$. Since there are $O(\tau n^2)$ start and merge vertices the lemma follows. We will discuss only the merge vertex case; the proof for a start vertex is the same.

Let v be a merge vertex, let $S \subset \mathcal{E}$ and $T \subset \mathcal{E}$ be the components that merge at v, and let p_a denote the path of entity $a \in S \cup T$ through $\mathcal{R}$, starting at v. The union over all a of these paths p_a forms a directed acyclic graph (DAG) $\mathcal{R}'_v$, which is a subgraph of $\mathcal{R}$ (see Figure 5.7 (a)). Consider "unraveling" $\mathcal{R}'_v$ into a tree $\mathcal{T}_v$ as follows. If p_a and p_b split in some vertex u and merge again in vertex w, with $t_w > t_u$ we duplicate the subpath starting at w. We duplicate these subpaths by decreasing order of time t_w. This yields a tree $\mathcal{T}_v$ with root v and at most $|S| + |T| \leq n$ leaves. Furthermore, all nodes in $\mathcal{T}_v$ have degree at most three (see Figure 5.7 (b)).

Since all maximal groups end at either a split or an end vertex, all maximal groups $G_1, .., G_k$ that start at v can now be represented by subpaths in $\mathcal{T}_v$ starting at the root. The path corresponding to a maximal group G ends at the first node where two entities $a, b \in G$ split, or at a leaf if no such node exists. Clearly, paths p_a and p_b can split only at a degree three node. Since $\mathcal{T}_v$ has at most n leaves it follows there are at most $O(n)$ degree three nodes.

Finally, we show that there is at most one maximal group that ends at a given leaf or degree three node of $\mathcal{T}_v$. Assume by contradiction that G_i and G_j, with $i \neq j$, both end at node u. Both maximal groups share the same path from the root of $\mathcal{T}_v$ to u, so all entities in G_i and G_j are in the same component at all times $t \in I = [t_v, t_u]$. Hence $G_i \cup G_j$ is a maximal group on I, contradicting that G_i and G_j were maximal. We conclude that the number of maximal groups k that start at v is at most the number of leaves plus the number of degree three nodes in $\mathcal{T}_v$. Hence $k = O(n)$. Summing over all $O(\tau n^2)$ start and merge vertices gives $O(\tau n^3)$ maximal groups in total. ☐

▶ Remark 5.6. It is easy to see that the proof of Theorem 5.5 actually gives us a bound of $O(|\mathcal{R}|n)$, where $|\mathcal{R}|$ denotes the complexity of the Reeb graph.

5.2 Computing the Grouping Structure

To compute the grouping structure we need to compute the reduced Reeb graph and the maximal groups. We now show how to do this efficiently. Removing the edges of the Reeb graph that are not used is an easy post-processing step which we do not discuss further.

5.2.1 Computing the Reeb Graph

We can compute the Reeb graph $\mathcal{R}$ as follows. We first compute all times where two entities a and b are at distance 2ε from each other. We distinguish two types of events, *connect events* at which a and b become directly connected, and *disconnect events* at which a and b stop being directly connected.

We now process the events on increasing time while maintaining the current components. We do this by maintaining a graph $G = (\mathcal{E}, Z)$ representing the directly-connected relation, and the connected components in G. The set of vertices in G is the set of entities. The graph G changes over time: at connect events we insert new edges into G, and at disconnect events we remove edges.

At any given time t, G contains an edge (a, b) if and only if a and b are directly connected at time t. Hence the components at t (the maximal sets of ε-connected entities) correspond to the connected components in G at time t. Since we know all times at which G changes in advance, we can use the same approach as Parsa [103] to maintain the connected components: we assign a weight to each edge in G and we represent the connected components using a maximum weight spanning forest. The weight of edge (a, b) is equal to the time at which we remove it from G, that is, the time at which a and b become directly disconnected. We store the maximum weight spanning forest F using a ST-tree [106], which allows for connectivity queries and updates in $O(\log n)$ time. We make sure that F is (partially-)persistent [44], so that we can represent the set of entities at each edge implicitly. This avoids having to store a copy of each entity in C_e at edge e.

We spend $O(n^2)$ time to initialize the graph G at t_0 in a brute-force manner. For each component we create a start vertex in $\mathcal{R}$. We also initialize a one-to-one mapping M from the current components in G to the corresponding vertices in $\mathcal{R}$. When we handle a connect event of entities a and b at time t, we query F to get the components C_a and C_b containing a and b, respectively. Using M we locate the corresponding vertices v_a and v_b in $\mathcal{R}$. If $C_a \neq C_b$ we create a new merge vertex v in $\mathcal{R}$ with time $t_v = t$, add edges (v_a, v) and (v_b, v) to $\mathcal{R}$ labeled C_a and C_b, respectively. If $C_a = C_b$ we do not change $\mathcal{R}$. Finally, we add the edge (a, b) to G (which may cause an update to F), and update the mapping M.

At a disconnect event we first query F to find the component C currently containing a and b. Using M we locate the vertex u corresponding to C. Next, we delete the edge (a, b) from G, and again query F. Let C_a and C_b denote the components containing a and b, respectively. If $C_a = C_b$ we are done, meaning a and b are still ε-connected. Otherwise we add a new split vertex v to $\mathcal{R}$ with time $t_v = t$, and an edge $e = (u, v)$ with $C_e = C$ as its component. We update M accordingly.

Finally, we add an end vertex v for each component C in F with $t_v = t_\tau$. We connect the vertex $u = M(C)$ to v by an edge $e = (u, v)$ and let $C_e = C$ be its component.

Analysis. We need $O(\tau n^2 \log n)$ time to compute all $O(\tau n^2)$ events and sort them according to increasing time. To handle an event we query F a constant number of times, and we insert or delete an edge in F. These operations all take $O(\log n)$ time. So the total time required for building $\mathcal{R}$ is $O(\tau n^2 \log n)$.

▶ **Theorem 5.7.** *Given a set $\mathcal{E}$ of n entities, in which each entity travels along a trajectory of τ edges, the Reeb graph $\mathcal{R}$ has $O(\tau n^2)$ vertices and edges, and can be computed in $O(\tau n^2 \log n)$ time.*

5.2.2 Computing the Maximal Groups

We now show how to compute all maximal groups using the Reeb graph $\mathcal{R} = (V, E)$. We will ignore the requirements that each maximal group should contain at least m entities and have a minimal duration of δ. That is, we assume $m = 1$ and $\delta = 0$. It is easy to adapt the algorithm for larger values.

Labeling the edges. Our algorithm labels each edge $e = (u, v)$ in the Reeb graph with a set of maximal groups $\mathcal{G}_e$. The groups $G \in \mathcal{G}_e$ are those groups for which we have discovered that G is a maximal group at a time $t \leq t_u$ and $G \subseteq C_e$. Each maximal group G becomes maximal at a vertex, either because a

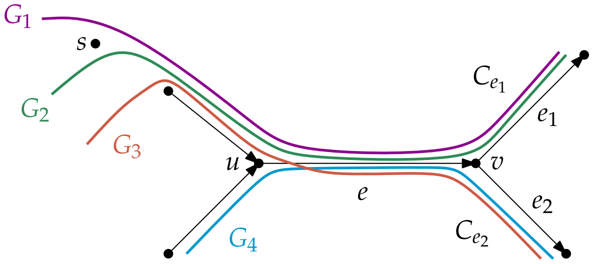


Figure 5.8: After split vertex v, $\mathcal{G}_{e_1}$ contains the groups $C_{e_1} = G_1 \cup G_2$ (with starting time t_s), G_1, and G_2. Maximal groups $C_{e_2} = G_3 \cup G_4$ (with starting time t_u), G_3, and G_4 go to e_2. The maximal groups C_e and $G_1 \cup G_2 \cup G_3$ end at v.

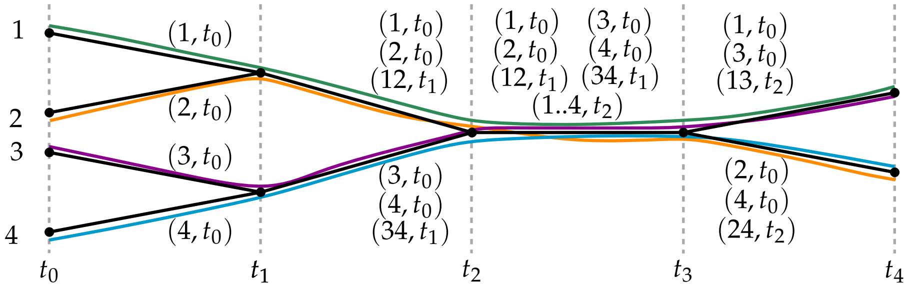


Figure 5.9: The maximal groups as computed by our algorithm (a set $\{i, j, k\}$ is denoted by ijk).

merge vertex created G as a new group that is maximal, or because G is now a maximal set of entities that is still together after a split vertex. This means we can compute all maximal groups as follows.

We traverse the set of vertices of $\mathcal{R}$ in topological order. For every vertex v we compute the maximal groups on its outgoing edge(s) using the information on its incoming edge(s).

If v is a start vertex it has one outgoing edge $e = (v, u)$. We set $\mathcal{G}_e$ to $\{(C_e, t_v)\}$ where $t_v = t_0$. If v is a merge vertex it has two incoming edges, e_1 and e_2. We propagate the maximal groups from e_1 and e_2 on to the outgoing edge e, and we discover (C_e, t_v) as a new maximal group. Hence $\mathcal{G}_e = \mathcal{G}_{e_1} \cup \mathcal{G}_{e_2} \cup \{(C_e, t_v)\}$.

If v is a split vertex it has one incoming edge e, and two outgoing edges e_1 and e_2. A maximal group G on e may end at v, continue on e_1 or e_2, or spawn a new maximal group $G' \subset G$ on either e_1 or e_2. In particular, for any group G' in $\mathcal{G}_{e_i}$, there is a group G in $\mathcal{G}_e$ such that $G' = G \cap C_{e_i} \neq \emptyset$. The starting time of G' is $t' = \min\{t \mid (G, t) \in \mathcal{G}_e \wedge G' \subseteq G\}$. Thus, t' is the first time G' was part of a maximal group on e. Stated differently, t' is the first time G' was in a component on a path to v. Figure 5.8 illustrates this case. If v is an end vertex it has no outgoing edges. So there is nothing to be done.

Figure 5.9 shows a complete example of a Reeb graph after labeling the edges with their maximal groups.

Storing the maximal groups. We need a way to store the maximal groups $\mathcal{G}_e$ on an edge $e = (u, v)$ in such a way that we can efficiently compute the set(s) of maximal groups on the outgoing edge(s) of a vertex v. We now show that we can use a tree $\mathcal{T}_e$ to represent $\mathcal{G}_e$, with which we can handle a merge vertex in $O(1)$ time, and a split vertex in $O(k)$ time, where k is the number of entities involved. The tree uses $O(k)$ storage.

We say a group G is a *subgroup* of a group H if and only if $G \subseteq H$ and $I_H \subseteq I_G$.

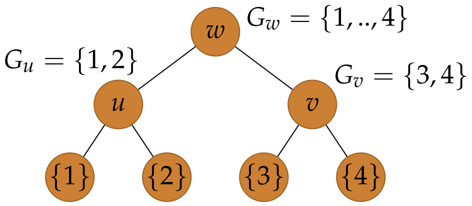


Figure 5.10: The grouping tree for the edge between t_2 and t_3 in Figure 5.9.

For example, in Figure 5.1(b) $\{a_1, a_2\}$ is a subgroup of $\{a_1, .., a_4\}$. Note that both G and H could be maximal.

▶ **Lemma 5.8.** *Let e be an edge of $\mathcal{R}$, and let S and T be maximal groups in $\mathcal{G}_e$ with starting times t_S and t_T, respectively. There is also a maximal group $G \supseteq S \cup T$ on e with starting time $t_G \geq \max(t_S, t_T)$, and if $S \cap T \neq \emptyset$ then S is a subgroup of T or vice versa.*

Proof. Clearly, $S, T \subseteq C_e$ and thus $S \cup T \subseteq C_e$. Component C_e itself is also a maximal group on e. By construction C_e must have the largest starting time t of the groups in $\mathcal{G}_e$. Hence $t_G \geq \max(t_S, t_T)$.

We prove the second statement by contradiction: assume $S \cap T \neq \emptyset$, and $S \not\subseteq T$ or vice versa. Assume w.l.o.g. that $t_S \leq t_T$. So the entities in S are all in a single component at all times $t \geq t_T \geq t_S$. At any time $t \geq t_T$ all entities in T are also in a single component. Since $S \cap T \neq \emptyset$ this must be the same component that contains S. Hence $S \subseteq T$, which together with $t_S \leq t_T$ proves the statement. ☐

We represent the groups $\mathcal{G}_e$ on an edge $e \in E$ by a tree $\mathcal{T}_e$ (see Figure 5.10). We call this the *grouping tree*. Each node v represents a group $G_v \in \mathcal{G}_e$. The children of a node v are the largest subgroups of G_v. From Lemma 5.8 it follows that any two children of v are disjoint. Hence an entity $a \in G_v$ occurs in only one child of v. Furthermore, note that the starting times are monotonically decreasing on the path from the root to a leaf: smaller groups started earlier. A leaf corresponds to a smallest maximal group on e: a singleton set with an entity $a \in C_e$. It follows that $\mathcal{T}_e$ has $O(n)$ leaves, and therefore has size $O(n)$. Note, however, that the summed sizes of all maximal groups can be quadratic.

Analysis. We analyze the time required to label each edge e with a tree $\mathcal{T}_e$ for a given Reeb graph $\mathcal{R}$. Topologically sorting the vertices takes linear time. So the running time is determined by the processing time in each vertex, that is, computing the tree(s) $\mathcal{T}_e$ on the outgoing edge(s) e of each vertex. We can handle all start vertices in $O(n)$ time in total. The end and merge vertices can be handled on $O(1)$ time each: the end vertices are trivial, and at a merge vertex v

the tree $\mathcal{T}_e$ is simply a new root node with time t_v and as children the (roots of the) trees of the incoming edges. At a split vertex we have to split the tree $\mathcal{T} = \mathcal{T}_{(u,v)}$ of the incoming edge (u, v) into two trees for the outgoing edges of v. For this, we traverse $\mathcal{T}$ in a bottom-up fashion, and for each node, check whether it induces a vertex in one or both of the trees after splitting. This algorithm runs in $O(|\mathcal{T}|)$ time. Since $|\mathcal{T}| = O(n)$ the total running time of our algorithm is $O(n|\mathcal{R}|) = O(\tau n^3)$.

Reporting the groups. We can augment our algorithm to report all maximal groups at split and end vertices. The main observation is that a maximal group ending at a split vertex v, corresponds exactly to a node in the tree $\mathcal{T}_{(u,v)}$ (before the split) that has entities in leaves below it that separate at v. The procedures for handling split and end vertices can easily be extended to report the maximal groups of size at least m and duration at least δ by simply checking this for each maximal group. Although the number of maximal groups is $O(\tau n^3)$ (Theorem 5.5), the summed size of all maximal groups can be $\Omega(\tau n^4)$. The running time of our algorithm is $O(\tau n^3 + N)$, where N is the total output size.

▶ **Theorem 5.9.** *Given a set $\mathcal{E}$ of n entities, in which each entity travels along a trajectory of τ edges, we can compute all maximal groups in $O(\tau n^3 + N)$ time, where N is the output size.*

▶ Remark 5.10. Once again we can formulate our result in terms of the size $|\mathcal{R}|$ of the Reeb graph $\mathcal{R}$: our algorithm runs in $O(\tau n^2 \log n + |\mathcal{R}|n + N)$ time where N is again the size of the output (which is now at most $O(|\mathcal{R}|n^2)$). The main difficulty in improving this is in getting an output-sensitive algorithm to construct $\mathcal{R}$.

5.3 Robustness

The grouping structure definition we have given and analyzed has a number of good properties. It fulfills monotonicity, and in the previous sections we showed that there are only polynomially many maximal groups, which can be computed in polynomial time as well. In this section we study the property of robustness, which our definition of grouping structure does not have yet. Intuitively, a robust grouping structure ignores short interruptions of groups, as these interruptions may be insignificant at the temporal scale at which we are studying the data. For example, if we are interested in groups that have a duration of one hour or more, we may want to consider interruptions of a minute or less insignificant.

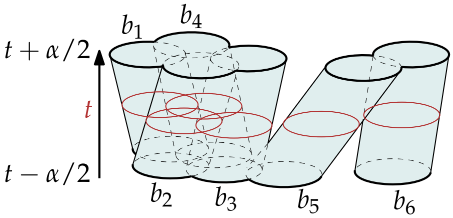


Figure 5.11: An α-component at time t.

We introduce a new temporal parameter α, which is related to the temporal scale at which the data is studied. Our robust grouping structure should ignore interruptions of duration at most α. We realize this by letting the precise moment of events be irrelevant beyond a value depending on α. Events that happen within α time of each other may cancel out, or their order may be exchanged. The objective is to incorporate α into our definitions while maintaining the properties that we have for the (non-robust) grouping structure. Note that α is another parameter that allows us to obtain more generalized views of the grouping structure by increasing its value. Obtaining generalized views in this way is related to the concept of persistence in computational topology [47, 49].

A possible definition of a robust grouping structure is based on the following intuition: A set of entities forms a robust group on I as long as every interval $I' \subseteq I$ on which its entities are not in the same component has length at most α. More formally: we say G is a *robust group* on time interval I if and only if: (*i*) G contains at least m entities, (*ii*) I has length at least δ, and (*iii*) for any time $t \in I$ there is a time $t' \in [t-\alpha/2,\ t+\alpha/2]$ and a component $C \in \boldsymbol{C}(t')$ such that $G \subseteq C$. Unfortunately, we can show that even determining whether there is a robust group of size k is NP-complete [28].

We consider a second definition for a robust group, which we will use from now on. Two entities are *α-relaxed directly connected* at time t if and only if they are directly connected at some time $t' \in [t-\alpha/2,\ t+\alpha/2]$. Two entities a and b are *α-relaxed ε-connected* at time t if there is a sequence $a = a_0, .., a_j = b$ such that a_i and a_{i+1} are α-relaxed directly connected. Note that the precise times may be different for different pairs a_i and a_{i+1}, as long as each time is in the interval $[t-\alpha/2,\ t+\alpha/2]$. A maximal set of α-relaxed ε-connected entities at time t is an *α-relaxed component*, or *α-component* for short. An α-component at time t corresponds to a connected $3D$-component in a horizontal slice of $\mathcal{M}$ with thickness α and centered at t (see Figure 5.11). This notion of α-component is similar to the α-Reeb graph as defined by Chazal and Sun [33].

A subset G of k entities is a *robust group* if and only if it is a group by the definition in the introduction, but where “component” is replaced by “α-component”

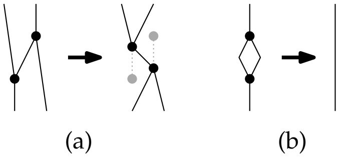


Figure 5.12: Passing encounter, before and after (a). Collapse encounter, before and after (b).

in condition (*iii*). This immediately leads to the definition of maximal robust groups and a robust grouping structure. The robust grouping structure has the property of monotonicity in the new parameter α as well. Note that every group which is a robust group according to the first definition, is also a robust group according to the second definition. The opposite is not true. For instance, in Figure 5.11, entities $b_1, .., b_6$ form a component by the second definition, but not by the first.

5.3.1 Computation of Maximal Robust Groups

We can compute all maximal robust groups according to the second definition. The idea is to modify the Reeb graph to a version that is parametrized by α and captures exactly the robust grouping structure for parameter α.

Let $\mathcal{R}$ be the Reeb graph that we used for the grouping structure without considering robustness. Note that this is the same as assuming $\alpha = 0$ in the definition of the robust grouping structure, and we let $\mathcal{R}_0 = \mathcal{R}$. For $\alpha > 0$ we define the Reeb graph parametrized in γ as $\mathcal{R}_\gamma$ by imagining a process that changes the Reeb graph for a growing parameter γ, starting with $\mathcal{R}_0$ and ending with $\mathcal{R}_{\alpha/2}$.

We observe that a new α-component starts at time $\alpha/2$ before two regular components merge and form a new component. Symmetrically, an α-component ends due to a split at time $\alpha/2$ after a regular component splits. Both facts follow from the new definition of α-relaxed directly connected. It implies that in the process that maintains $\mathcal{R}_\gamma$ for growing γ, the split nodes move forward in time, zippering together the outgoing edges, and the merge nodes move backward in time, zippering together the incoming edges. All nodes move at the same rate in γ, which implies that in the process, the only event where the Reeb graph changes structurally is when an (earlier) split node encounters a (later) merge node. This can happen only if they are endpoints of the same edge of the Reeb graph. The encounter is either a *passing* or a *collapse* (see Figure 5.12).

Both encounters lead to new edges in the Reeb graph and can thus give rise to new encounters when growing γ further. The collapse encounter reduces the complexity of the Reeb graph: two nodes of degree 3 disappear and four

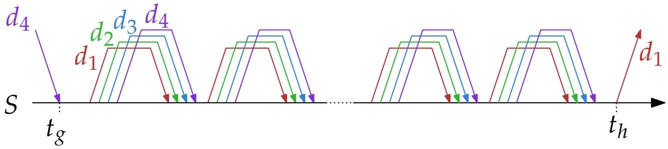


Figure 5.13: The part of the Reeb graph that yields $\Omega(n^3)$ encounter events (for $n = 16$).

edges become a single edge. The collapse event is exactly the situation where a component splits and merges again, so by removing a split-merge pair involving the same entities we ignore the temporary split of a component (or group).

A passing encounter maintains the complexity of the Reeb graph. Before the passing encounter, a part of one group splits and merges with a different group. After the passing encounter, the two groups merge (for a short time) and then split again.

Next, we show that there are $O(\tau n^3)$ encounter events in the Reeb graph of the robust version of the trajectory grouping structure, and this bound is tight in the worst case.

▶ **Lemma 5.11.** *For some set $\mathcal{E}$ of n entities, in which each entity travels along a trajectory of τ edges, the structure of the Reeb graph $\mathcal{R}_\gamma$ of $\mathcal{E}$ changes $\Omega(\tau n^3)$ times when increasing γ from zero to infinity.*

Proof. We show that there is a set of n trajectories, each consisting of τ edges, for which there are $\Omega(\tau n^3)$ encounter events. The lemma then follows.

We use the same construction as in Lemma 5.4. So in all time intervals $[t_{2i}, t_{2i+1}]$ we have a set S of $3n/4$ stationary entities/discs and a set $D = \{d_1, .., d_{n/4}\}$ entities, ordered from right to left, that move to the right in such a way that d_i becomes directly (dis)connected with S before d_{i+1} (see Figure 5.5). Let t_g be the first time at which $d_{n/4}$ becomes directly connected with S, and let t_h denote the last time d_1 becomes directly disconnected with S. We now show that the part of Reeb graph $\mathcal{R}'$ corresponding to the interval (t_g, t_h) already yields $\Omega(n^3)$ encounter events. Note that no other encounter events involving other parts of the Reeb graph can interfere with the encounter events in $\mathcal{R}'$.

In between t_g and t_h every disc d_i becomes directly (dis)connected with S $\Omega(n)$ times. So $\mathcal{R}'_\gamma$ initially contains of a path P of $\Omega(n^2)$ edges. Each edge has at least the set of entities S associated with it, and possibly other entities as well. The vertices on P can be grouped in $\Omega(n)$ sequences of $k = n/4$ split vertices $u_1, .., u_k$ followed by k merge vertices $v_1, .., v_k$. At vertex u_i entity d_i splits from S and at v_i entity d_i merges with S. See Figure 5.13.

By increasing γ each split vertex u_i will have a passing encounter with the merge vertices $v_1, .., v_{i-1}$ before it collapses with v_i. Hence each sequence involves $\sum_{i=1}^{k}(i-1) = \Omega(n^2)$ encounter events. Since there are $\Omega(n)$ such

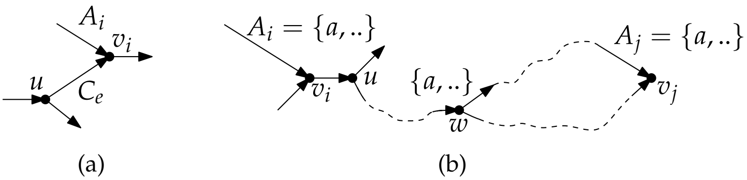


Figure 5.14: The part of $\mathcal{R}_\gamma$ before u encounters v_i. The set A_i merges with C_e at vertex v_i (a). If a merges at both v_i and v_j it has to leave (split) at a vertex w in between (b).

sequences this gives $\Omega(n^3)$ encounter events in a single timestep, and hence $\Omega(\tau n^3)$ in total. □

▶ **Theorem 5.12.** *Let $\mathcal{E}$ be a set of n entities, in which each entity travels along a trajectory of τ edges. The structure of the Reeb graph $\mathcal{R}_\gamma$ of $\mathcal{E}$ changes at most $O(\tau n^3)$ times when increasing γ from zero to infinity. This bound is tight in the worst case.*

Proof. Lemma 5.11 gives a construction that shows that there may be $\Omega(\tau n^3)$ encounters.

Since each collapse event decreases the number of edges by three it follows the number of collapse events is at most $O(\tau n^2)$. What remains is to prove that the number of passing events is $O(\tau n^3)$. Each passing event involves a split vertex u and a merge vertex v. We now show that there are at most n passing events involving a given split vertex u. Since there are $O(\tau n^2)$ split vertices this means the number of passing events is at most $O(\tau n^3)$.

Assume by contradiction that there are $k > n$ passing events involving split vertex u. Let $\gamma_1, .., \gamma_k$ be the values for γ for which these passing events occur in non-decreasing order, and let $v_1, .., v_k$ be the corresponding merge vertices. Just before u passes v_i the edge $e = (u, v_i)$ is an incoming edge of v_i. Let A_i denote the set of entities on the other incoming edge of v_i, that is the set of entities that merges with C_e at vertex v_i (see Figure 5.14(a)).

Since $k > n$ there must be an entity a that u "passes" at least twice. That is, u passes v_i and v_j, with $i < j$, and $a \in A_i$ and $a \in A_j$. Now consider the Reeb graph $\mathcal{R}_\gamma$ just after u passes v_i (which means $\gamma > \gamma_i$). Since u still has to pass v_j there is a path Q connecting u to v_j. By further increasing γ this path will eventually become a single edge (u, v_j), which will flip to (v_j, u) when u passes v_j at $\gamma = \gamma_j$.

Entity a is present at the first vertex of Q (vertex u), and it merges again with path Q at v_j. Clearly, this means that Q contains a split vertex w at which a splits from path Q before it can return to Q in vertex v_j (see Figure 5.14 (b)).

We now have two paths connecting w to v_j: the path that a follows and the subpath of Q. We again have that by increasing γ both paths will become singleton edges connecting w to v_j. Eventually both these edges are removed in a collapse event for some $\hat{\gamma}$. If $w = u$ this means (u, v_j) is actually a collapse event instead of a passing event. Contradiction. If $w \neq u$ we have that $t_w > t_u$, and therefore $\hat{\gamma} < \gamma_j$. The collapse event at $\hat{\gamma}$ will consume both w and v_j, which means u can no longer pass v_j. Contradiction. We conclude that the number of passing events involving u is at most n. With $O(\tau n^2)$ vertices this yields the desired bound of $O(\tau n^3)$ passing events. $\square$

Algorithmically, we start with the Reeb graph $\mathcal{R}_0$ and examine each edge. Any edge that leads from a split node to a merge node and whose duration is at most α is inserted in a priority queue, where the duration of the edge is the priority. We handle the encounter events in the correct order, changing the Reeb graph and possibly inserting new encounter events in the priority queue. Each event is handled in $O(\log n)$ time since it involves at most $O(1)$ priority queue operations. Since there are $O(\tau n^3)$ events (Theorem 5.12) this takes $O(\tau n^3 \log n)$ time in total. Once we have the Reeb graph $\mathcal{R}_{\alpha/2}$, we can associate the trajectories with its edges as before. The computation of the maximal robust groups is done in the same way as computing the maximal groups on the normal Reeb graph $\mathcal{R}$. We conclude:

▶ **Theorem 5.13.** *Given a set $\mathcal{E}$ of n entities, in which each entity travels along a trajectory of τ edges, we can compute all robust maximal groups in $O(\tau n^3 \log n + N)$ time, where N is the output size.*

5.4 Evaluation

To see if our model of the grouping structure is practical and indeed captures the grouping behavior of the entities we implemented and evaluated our algorithms. We would like to visually inspect the maximal groups identified by our algorithm, and compare this to our intuition of groups. For a small number of (short) trajectories we can still show this in a figure, see for example Figure 5.15, which shows the monotonicity of the maximal groups in size and duration. However, for a larger number of trajectories the resulting figures become too cluttered to analyze. So instead we generated short videos.[1]

We use two types of data sets to evaluate our method: a synthetic data set generated using a slightly modified version of the NetLogo Flocking model [119, 120], and a real-world data set consisting of deer, elk, and cattle, tracked in the Starkey project [100].

---

[1] See `www.staff.science.uu.nl/~staal006/grouping`.

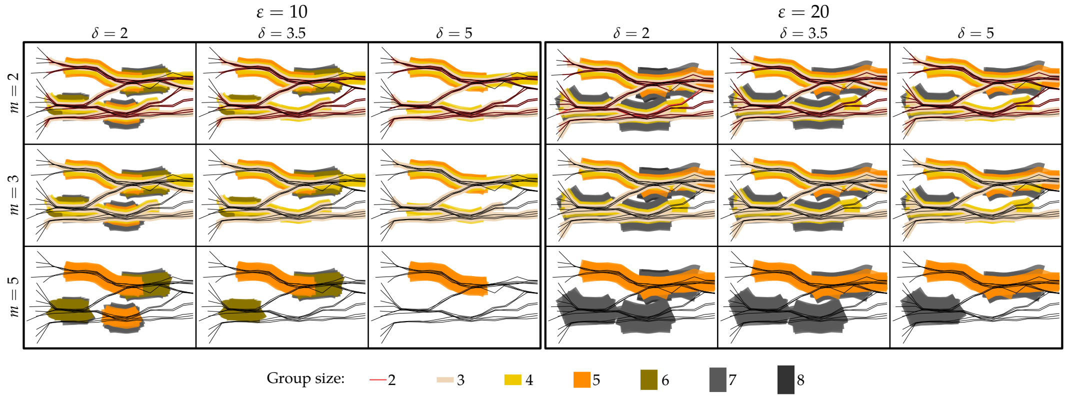


Figure 5.15: The maximal groups for varying parameter values. The time associated with each trajectory vertex is proportional to its x-coordinate.

NetLogo. We generated several data sets using an adapted version of the NetLogo Flocking model [120]. In our adapted model the entities no longer wrap around the world border, but instead start to turn when they approach the border. Furthermore, we allow small random direction changes for the entities. The data set that we consider here contains 400 trajectories, with 818 edges each. Similar to Figure 5.15, our videos show all maximal groups for varying parameter values.

The videos show that our model indeed captures the crucial properties of grouping behavior well. We notice that the choice of parameter values is important. In particular, if we make ε too large we see that the entities are loosely coupled, and too many groups are found. Similarly, for large values of m virtually no groups are found. However, for reasonable parameter settings, for example $\varepsilon = 5.25$, $m = 4$, and $\delta = 100$, we can see that our algorithm identified virtually all sets of entities that travel together. Furthermore, if we see a set of entities traveling together that is not identified as group, we indeed see that they disperse quickly after they have come together. The coloring of the line-segments also nicely shows how smaller groups merge into larger ones, and how the larger groups break up into smaller subgroups. This is further evidence that our model captures the grouping behavior well.

Starkey. We also ran our algorithms on a real-world data set, namely on tracking data obtained in the Starkey project [100]. This data set captures the movement of deer, elk, and cattle in Starkey, a large forest area in Oregon (US), over three years. Not all animals are tracked during the entire period, and positions are not reported synchronously for all entities. Thus, we consider only a subset of the data, and resample the data such that all trajectories have vertices at the same (regularly spaced) times. We chose a period of 30 days for which we have the locations of most of the animals. This yields a data set containing 126 trajectories with 1264 vertices each. In the Starkey video we can see that a large group of entities quickly forms in the center, and then slowly splits into multiple smaller groups. We notice that some entities (groups) move closely together, whereas others often stay stationary, or travel separately.

Running times. Since we are mainly interested in how well our model captures the grouping behavior, we do not extensively evaluate the running times of our algorithms. On our desktop system with a AMD Phenom II X2 CPU running at 3.2Ghz our algorithm, implemented in Haskell, computes the grouping structure for our data sets in a few seconds. Even for 160 trajectories with roughly 20 thousand vertices each we can compute and report all maximal groups in three minutes. Most of the time is spent on computing the Reeb graph, in particular on computing the connect/disconnect events. Our implementation uses a slightly easier, yet slower, data structure to represent the maximum-weight spanning forest during the construction of the Reeb graph compared to the ST-trees described in Section 5.2.1. So we expect that some speedup is still possible.

5.5 Concluding Remarks

We introduced a trajectory grouping structure which uses Reeb graphs and a notion of persistence for robustness. We showed how to characterize and efficiently compute the maximal groups and group changes in a set of trajectories, and bounded their maximal number. Our paper demonstrates that computational topology provides a mathematically sound way to define grouping of moving entities. The complexity bounds, algorithms and implementation together form the first comprehensive study of grouping. Our videos show that our methods produce results that correspond to human intuition.

Throughout this chapter, we assumed that the entities move in $\mathbb{R}^2$. Note however, that our analysis and our algorithms do not actually use, or need, this information. Indeed, they use only the structure of the Reeb graph. Furthermore, to construct the Reeb graph we need only the times at which two entities are at distance ε. There are only $O(\tau n^2)$ such times, even for entities moving in $\mathbb{R}^d$. Hence, all our results also hold for entities moving in $\mathbb{R}^d$, with $d > 2$.

Further work includes more extensive experiments together with domain specialists, such as behavioral biologists, to ensure further that the grouping structure captures groups and events in a natural, expected way, and changes in the parameters have the desired effect. At the same time, our research may be linked to behavioral models of collective motion [110] and provide a (quantifiable) comparison of these.

We expect that for realistic inputs the size of the Reeb graph, and thus the grouping structure, is much smaller than the worst-case bound that we proved. Hence an interesting open problem is to design an output-sensitive algorithm to construct the Reeb graph, or to analyze its complexity under realistic input models.

Chapter 6

Grouping under Geodesic Distance

In this chapter we extend the trajectory grouping structure from the previous chapter by incorporating contextual information. The entities generating the trajectories typically do not move in an infinitely large unrestricted space. Instead, they live in a world containing buildings, walls, lakes, etc., through which they cannot move. We incorporate obstacles into the trajectory grouping structure, and measure the distance between entities by their *geodesic distance*. The geodesic distance between two entities is the distance that needs to be traversed for one entity to reach the other entity. This approach gives a more natural notion of groups because it separates entities moving on opposite sides of obstacles like fences or water bodies.

Trajectory grouping structure. We are given a set $\mathcal{E}$ of n entities, each moving along a piecewise linear trajectory with τ vertices, and a set of pairwise disjoint polygonal obstacles $\mho = \{\mathcal{O}_1, .., \mathcal{O}_h\}$. Let m denote the total complexity of $\mho$. Our objective is again to compute the trajectory grouping structure and all maximal groups. To this end we need the Reeb graph $\mathcal{R}$ capturing connectivity events of the entities in $\mathcal{E}$ (see Section 5.1). Since the entities now move amidst obstacles we measure the distance between two entities and by their geodesic distance. This means that we may get more than $O(\tau n^2)$ times at which two sets of entities are at distance ε. We refer to such times as *critical events*. Since the Reeb graph has a vertex for each critical event, it may also have a complexity higher than $O(\tau n^2)$. Furthermore, to construct $\mathcal{R}$ we need to find those critical events. It is no longer obvious how to do that efficiently.

Once we have the Reeb graph $\mathcal{R}$, we can use the analysis from Section 5.1.2 to bound the number of maximal groups, and the algorithms from Section 5.2 to compute them. So the interesting part is to analyze the complexity of $\mathcal{R}$ and to determine how to compute it.

Terminology and notation. Recall that we denote the position of entity a at time t by $a(t)$, and that $\|pq\|$ denotes the Euclidean distance between points p and q. Let $\xi_{ab}(t) = \|a(t)b(t)\|$ denote the (Euclidean) distance between entities a

and b at time t. A *path* $P = p_1, .., p_k$ from p_1 to p_k is a polygonal line with vertices $p_1, .., p_k$, and has length $\varsigma(P) = \sum_{i=1}^{k-1} \|p_i p_{i+1}\|$. A path is *obstacle-avoiding* if it is disjoint from the interior of all obstacles in $\mathcal{O}$. A path between p and q is a *geodesic*, denoted $g(p, q)$, if it has minimum length among all obstacle-avoiding paths. We refer to the length of $g(p, q)$ as the *geodesic distance* between p and q. We denote the geodesic distance between a and b at time t by $\varsigma_{ab}(t) = \varsigma(g(a(t), b(t)))$.

To determine if a set of entities forms a group, we have to decide if they are close together. As in Chapter 5 we model this by a spatial parameter ε: two entities a and b are *directly connected* at time t if they are within (geodesic) distance ε from each other, that is, $\varsigma_{ab}(t) \leq \varepsilon$. A set of entities $\mathcal{E}'$ is *ε-connected* at time t if for any pair $a, b \in \mathcal{E}'$ there is a sequence $a = a_0, a_1, .., a_k = b$ such that a_i and a_{i+1} are directly connected.

We refer to a time at which entities a and b become directly connected or disconnected as an *ε-event*. At such a time the distance between a and b is exactly ε. If an ε-event also connects or disconnects the maximal ε-connected set(s) containing a and b, it is a *critical event*. A (maximal) ε-connected set of entities $\mathcal{E}'$ is a *group* if it is ε-connected at any time t in a time interval of length at least δ, and it has at least a certain size. We are again interested only in maximal groups (see Chapter 5).

Results and organization. The number of critical events, and thus the size of the Reeb graph, depends on the obstacles and their complexity. We study three settings for the obstacles. In the simplest case, all entities move inside a simple polygon with m vertices. In the most general case, individual obstacles can have any shape, location, and complexity, but they are disjoint and have total complexity m. As an intermediate case we assume that the distance between any two non-adjacent obstacle edges is at least ε. In this case we say that the obstacles are *well-spaced*.

Our results are listed in Table 6.1. For the simple polygon case, which we treat in Section 6.2, our bounds are tight. The upper bounds for the well-spaced obstacles case, and the general obstacles case include a $\lambda_4(n)$ term, where $\lambda_s(n)$ denotes the maximum length of a Davenport-Schinzel sequence of order s with n symbols. Since $\lambda_4(n)$ is only slightly superlinear, our bounds for these cases are almost tight. We present these results in Sections 6.3 and 6.4, respectively. For all cases we also bound the total number of ε-events, and we show how to compute $\mathcal{R}$ efficiently.

	Lower bound	Upper bound	Algorithm
No obstacles	$\Omega(\tau n^2)$	$O(\tau n^2)$	$O(\tau n^2 \log n)$
Simple polygon	$\Omega(\tau n^2)$	$O(\tau n^2)$	$O(\tau n^2(\log^2 m + \log n) + m)$
Well-spaced obstacles	$\Omega(\tau(n^2 + nm))$	$O(\tau(n^2 + m\lambda_4(n)))$	$O(\tau n^2 m \log n)$
General obstacles	$\Omega(\tau(n^2 + nm \min\{n, m\}))$	$O(\tau(n^2 + m^2\lambda_4(n)))$	$O(\tau n^2 m^2 \log n + m^2 \log m)$

Table 6.1: The number of critical events (i.e. the size of $\mathcal{R}$), and the time required to construct $\mathcal{R}$. Note that the input size is $\Theta(\tau n + m)$ (with $m = 0$ in the "No obstacles" case).

6.1 Preliminaries

Let a and b be two entities, each moving with a constant speed along a straight line during interval I, and let p be a fixed point in $\mathbb{R}^2$. During I, the Euclidean distance $\xi_{ap}(t)$ between a and p is a convex hyperbolic function in t that has the form $\sqrt{Q(t)}$, for some quadratic function Q. The Euclidean distance between a and b is a convex hyperbolic function of the same form. Since ξ_{ap} is convex, there are at most two times in I such that $\xi_{ap}(t) = \varepsilon$. The same applies for ξ_{ab}.

The geodesic distance $\varsigma_{ap}(t)$ between a and p is a piecewise function. At times where the geodesic $g(a(t), p)$ consists of a single line segment, the geodesic distance is simply the Euclidean distance. When the geodesic consists of more than one line segment we can decompose it into two parts: a line segment $g(a(t), u) = \overline{a(t)u}$, and a path $g(u, p)$, where u is the first obstacle vertex on $g(a(t), p)$. Similarly, if the geodesic $g(a(t), b(t))$ between a and b consists of more than one segment we can decompose it into three parts $\overline{a(t)u}$, $g(u, v)$, and $\overline{vb(t)}$ (we may have $u = v$). It follows that each piece of ς_{ap} is convex and hyperbolic. The pieces of ς_{ab} are convex as well, since they are of the form $\xi_{au}(t) + C + \xi_{vb}(t) = \sqrt{Q_1(t)} + C + \sqrt{Q_2(t)}$, for some quadratic functions Q_1 and Q_2 and a constant C. Therefore, we again have that on each piece there are at most two times where $\varsigma_{ap}(t)$ is exactly ε. The same applies for $\varsigma_{ab}(t)$.

We obtain the same results when a and b move on piecewise linear trajectories, rather than lines. The functions then simply consist of more pieces.

▶ **Lemma 6.1.** *Let* $\mathcal{F} = f_1, .., f_n$ *be a set of n piecewise (partial) functions, each function* f_i *consisting of* τ *pieces* $f_i^1, .., f_i^\tau$*, such that any two pieces* f_i^k *and* f_j^ℓ *intersect each other at most s times. The lower envelope* $\mathcal{L}$ *of* $\mathcal{F}$ *has complexity* $O(\tau\lambda_{s+2}(n))$.

Proof. Let $D = \bigcup \mathrm{dom}(f_i)$ denote the domain of $\mathcal{F}$. We partition D into τ intervals $D_1, .., D_\tau$, such that each interval contains at most $O(n)$ pieces of functions in $\mathcal{F}$; if a piece f_i^k is (partially) defined in two subsequent intervals D_ℓ and $D_{\ell+1}$ we split f_i^k into two separate pieces. The total number of pieces in $\mathcal{F}$ is still $O(\tau n)$, and the number of pieces in a single interval D_i is also still $O(n)$. Thus, in each interval D_i, the lower envelope $\mathcal{L}_i$ of the pieces in that lie in D_i has complexity at most $O(\lambda_{s+2}(n))$ [2]. The intervals D_i are disjoint, so the lower envelope of $\mathcal{F}$ is the concatenation of the τ lower envelopes $\mathcal{L}_i$. The lemma follows. ☐

Analogous to Lemma 6.1 we can show that the upper envelope of $\mathcal{F}$ has complexity $O(\tau\lambda_{s+2}(n))$.

6.2 Simple Polygon

We first focus our attention on entities moving in a simply-connected polygonal domain.

6.2.1 Lower Bound

For entities moving in $\mathbb{R}^2$ without obstacles the number of critical events can be $\Omega(\tau n^2)$ (Lemma 5.1). Clearly, this lower bound also holds for entities moving inside a simple polygon.

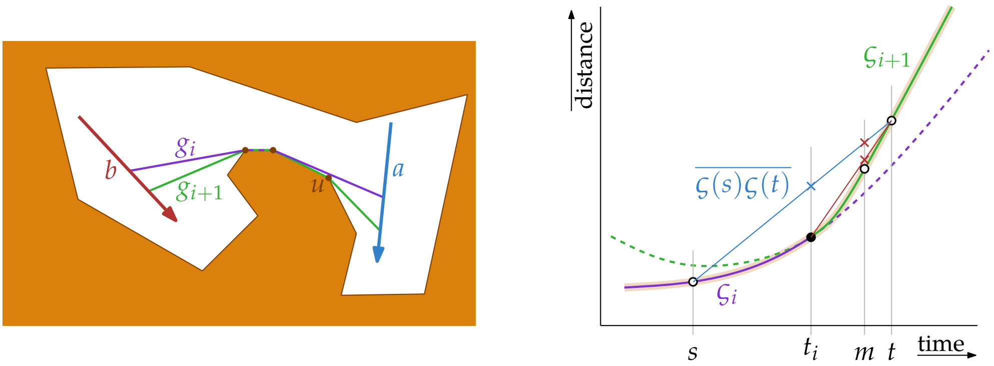
6.2.2 Upper Bound

Let a and b be two entities, each moving along a line during interval I, and let $\varsigma(t) = \varsigma_{ab}(t)$ be the function describing the geodesic distance between a and b during interval I.

▶ **Lemma 6.2.** *The function ς is convex.*

Proof. Let $[t_{i-1}, t_i]$ and $[t_i, t_{i+1}]$ be two consecutive time intervals, corresponding to pieces ς_i and ς_{i+1} of ς. We now show that ς is convex on $[t_{i-1}, t_{i+1}]$.

Let g_i and g_{i+1} denote the geodesic shortest paths corresponding to ς_i and ς_{i+1}, respectively. Geodesics g_i and g_{i+1} differ by at most one vertex u (assuming general position of the obstacle vertices), and this vertex occurs either at the beginning or the end of the geodesic. Consider the case that u is the first vertex of g_{i+1}, and u does not occur on g_i. See Figure 6.1(a). All other cases are symmetric. Let v be the second vertex of g_{i+1} (and thus the first vertex of g_i). We have $\varsigma_i(t) = \|a(t)v\| + \varsigma(v, b(t))$ and $\varsigma_{i+1}(t) = \|a(t)u\| + \|uv\| + \varsigma(v, b(t))$. It



(a) (b)

Figure 6.1: (a) Geodesics g_i (purple) and g_{i+1} differ by at most one vertex; the first vertex u on g_{i+1}. (b) The corresponding functions ς_i and ς_{i+1}. Using that $\varsigma(t_i)$ lies below $\overline{\varsigma(s)\varsigma(t)}$ (blue line segment) and that $\varsigma(m)$ lies below $\overline{\varsigma(t_i)\varsigma(t)}$ (red line segment) we show that that ς is convex.

follows that the individual pieces ς_i and ς_{i+1} are (convex) hyperbolic functions, that $\varsigma_i(t_i) = \varsigma_{i+1}(t_i)$, and that for any time $t \in [t_{i-1}, t_{i+1}]$, $\varsigma_{i+1}(t) \geq \varsigma_i(t)$. We use these properties to show that ς is convex on $[t_{i-1}, t_{i+1}]$.

Consider the space $\mathbb{T} \times \mathbb{R}$. We show that for any three times $s, m, t \in [t_{i-1}, t_{i+1}]$, with $s \leq m \leq t$, the point $\varsigma(m)$ lies below the line segment (function) $\overline{\varsigma(s)\varsigma(t)}$, that is $\varsigma(m) \leq \overline{\varsigma(s)\varsigma(t)}(m)$. Since ς_i and ς_{i+1} are convex, the only interesting case is when s lies on ς_i and t lies on ς_{i+1}. We proceed by case distinction on m.

For $m \leq t_i$ we use that ς_i is convex and that that for any time t, $\varsigma_{i+1}(t) \geq \varsigma_i(t)$:

$$\varsigma(m) = \varsigma_i(m) \leq \overline{\varsigma_i(s)\varsigma_i(t)}(m) \leq \overline{\varsigma_i(s)\varsigma_{i+1}(t)}(m) = \overline{\varsigma(s)\varsigma(t)}(m).$$

For $m > t_i$ we use that ς_{i+1} is convex, and thus $\varsigma(m) \leq \overline{\varsigma_{i+1}(t_i)\varsigma_{i+1}(t)}(m) = \overline{\varsigma(t_i)\varsigma(t)}(m)$. By using the same argument as for the previous case we get $\varsigma(t_i) \leq \overline{\varsigma(s)\varsigma(t)}(t_i)$. It follows that the line segment $\overline{\varsigma(t_i)\varsigma(t)}$ lies below $\overline{\varsigma(s)\varsigma(t)}$, and thus $\varsigma(m) \leq \overline{\varsigma(t_i)\varsigma(t)}(m) \leq \overline{\varsigma(s)\varsigma(t)}(m)$. See Figure 6.1(b) for an illustration. It follows that ς is convex. ☐

▶ **Theorem 6.3.** *Let $\mathcal{E}$ be a set of n entities, each moving in a simple polygon along a piecewise linear trajectory with τ vertices. The number of ε-events is at most $O(\tau n^2)$.*

Proof. Fix a pair of entities a and b. Both a and b move along trajectories with τ vertices. So there are $2\tau - 1$ intervals during which both a and b move along a line. During each such interval ς_{ab} is convex (Lemma 6.2). So there are at most two times in each interval at which $\varsigma_{ab}(t) = \varepsilon$. The lemma follows. ☐

6.2.3 Algorithm

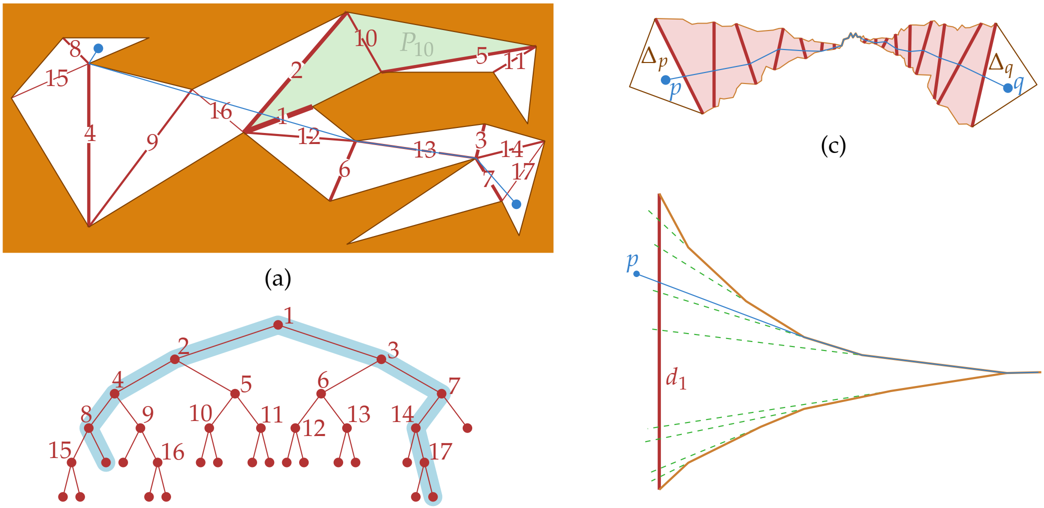
Next, we describe how to compute all ε-events. The high level overview of our algorithm is as follows. For each pair of entities a and b, we first find a time $t_{\min}$ such that the geodesic distance $\varsigma(t) = \varsigma_{ab}(t)$ between a and b is minimal. Clearly, if $\varsigma(t_{\min}) > \varepsilon$ there is no time at which a and b are at distance ε. Otherwise, we use the fact that ς is convex (Lemma 6.2). This means that on $I^- = (-\infty, t_{\min}]$ it is monotonically decreasing, and on $I^+ = [t_{\min}, \infty)$ it is monotonically increasing. Hence, there are at most two times $t^- \in I^-$ and $t^+ \in I^+$ such that $\varsigma(t) = \varepsilon$. We now find t^- and t^+ using parametric search: t^- (t^+) is the earliest (latest) time in I^- (I^+) such that $\varsigma(t) \leq \varepsilon$. To actually find $t_{\min}$, we use the same approach. At $t_{\min}$ the derivative ς' of ς is zero. Since ς is convex, its derivative is monotonically increasing. Therefore, we can find $t_{\min}$ using a parametric search: $t_{\min}$ is the earliest time such that $\varsigma'(t) \geq 0$.

Finding the times $t_{\min}$, t^-, and t^+. We use the generic parametric search algorithm described in Section 2.1. For our purpose, it suffices to use only the first part of the technique, i.e. the result from Theorem 2.1.

To find $t_{\min}$ we use $\mathcal{P}(t) = \varsigma'(t) \geq 0$ as predicate. To find t^- and t^+ we use $\mathcal{P}(t) \leq \varepsilon$, and $\mathcal{P}(t) \geq \varepsilon$, respectively. In all these cases we need an algorithm $\mathcal{A}$ that can test $\mathcal{P}(t)$ for a given time t. This means that we need an efficient algorithm to compute $\varsigma(t)$ and a functional description of ς. To this end, we preprocess the input polygon for shortest path queries. We triangulate the polygon in $O(m)$ time [34], and build the data structure $\mathcal{D}$ of Guibas and Hershberger [64]. This also takes $O(m)$ time, and allows us to find the length of the shortest path between two fixed points p and q in $O(\log m)$ time. In particular, this means that for a given time t, we can compute $\varsigma(t)$ and $\varsigma'(t)$ in $O(\log m)$ time.

In the parametric search we have to run our algorithm $\mathcal{A}$ on the unknown optimum t^*, with the goal of generating comparisons $E(t^*)$. This means we have to query $\mathcal{D}$ with the unknown points $a(t^*)$ and $b(t^*)$. Before we describe this process in more detail we briefly review the data structure $\mathcal{D}$ of Guibas and Hershberger [64].

Data structure $\mathcal{D}$ represents the polygon P by a balanced binary tree. Each node ν in the tree represents a sub-polygon P_ν of P, and stores a diagonal that splits P_ν into two sub-polygons; the polygons corresponding to the children of ν. The leaves of the tree represent triangles. See Figure 6.2(a) and (b). Additionally, each node ν stores a set of *hourglasses*. An hourglass H_{ef} represents the shortest paths between two diagonals e and f of P_ν. Two such hourglasses H_{ef} and H_{fg} can be concatenated (combined) into a new hourglass H_{eg} efficiently [64].



(a)

(b)

(c)

(d)

Figure 6.2: (a) Input polygon P is decomposed into sub-polygons by its diagonals. (b) The binary tree corresponding to the diagonals. A shortest path between p and q corresponds to a sequence of diagonals in the tree. (c) Each sub-polygon stores a set of hourglasses that represent shortest paths between diagonals. These hourglasses can be concatenated together. (d) The final step of a shortest path query: finding the segment through which the shortest path enters the hourglass.

A shortest path query in $\mathcal{D}$ with end points p and q consists of four steps: (1) We find the triangles Δ_p and Δ_q containing p and q, respectively, using the point location data structure by Edelsbrunner, Guibas, and Stolfi [46]. (2) We find the sequence of diagonals (nodes) $d_1, .., d_k$ on the path between Δ_p and Δ_q in $\mathcal{D}$. (3) We concatenate the hourglasses on this path to obtain one hourglass H between d_1 and d_k. See Figure 6.2(c). (4) The hourglass H partitions d_1 and d_k into small segments. We need to find the segment through which the shortest path between p and q enters H and through which it leaves H. See Figure 6.2(d).

A shortest path query at the unknown time t^*. Consider a query in $\mathcal{D}$ with the unknown points $a(t^*)$ and $b(t^*)$. In step (1) we query the data structure by Edelsbrunner, Guibas, and Stolfi [46] to find the triangle $\Delta_{a(t^*)}$ containing $a(t^*)$. In such a query, we follow a path in a layered DAG. Each node in this DAG corresponds to a comparison of the form: "is $a(t^*)_x$ at most x", or "is $a(t^*)$ above line segment s". Hence, each such node yields a comparison (low degree polynomial) involving t^*. Similarly, we find the triangle $\Delta_{b(t^*)}$ containing $b(t^*)$. Once we know these triangles, steps (2) and (3) of the query algorithm are independent of t^*; that is, they do not yield any new comparisons involving t^*. Finally, in step (4) we determine through which segments the shortest path between $a(t^*)$ and $b(t^*)$ enters and leaves hourglass H. This involves computing the intersection points between d_1 (d_k) and a number of line segments $\overline{a(t^*)h_i}$ ($\overline{b(t^*)h_i}$), for some $h_i \in H$. Each such intersection yields a small number of comparisons (low degree polynomials) involving t^*. By now we have encountered all comparisons made by the query algorithm, so we can obtain the exact value of t^* from the interval I^*. Furthermore, note that H provides the description of the piece of the function ς containing $\varsigma(t^*)$ that we need.

Since a query, and thus our algorithm $\mathcal{A}$, takes $O(\log m)$ time, it now immediately follows from Theorem 2.1 that we can compute $t_{\min}$, t^-, and t^+ in $O(\log^2 m)$ time each. We thus obtain the following result.

▶ **Lemma 6.4.** *Let $\mathcal{E}$ be a set of n entities, each moving in a simple polygon $\mathcal{P}$ along a piecewise linear trajectory with τ vertices. We can compute all ε-events in $O(\tau n^2 \log^2 m + m)$ time, where m is the number of vertices in $\mathcal{P}$.*

To compute $\mathcal{R}$ we can now use the algorithm from Theorem 5.7: at each ε-event we insert or delete an edge in G. This takes $O(\log n)$ time per ε-event, and thus $O(\tau n^2 \log n)$ time in total. We conclude:

▶ **Theorem 6.5.** *Let $\mathcal{E}$ be a set of n entities, each moving in a simple polygon $\mathcal{P}$ along a piecewise linear trajectory with τ vertices. The Reeb graph $\mathcal{R}$ representing the movement of the entities in $\mathcal{E}$ has size $O(\tau n^2)$ and can be computed in $O(\tau n^2(\log^2 m + \log n) + m)$ time, where m is the number of vertices in $\mathcal{P}$.*

6.3 Well-spaced Obstacles

Next, we consider the situation where the entities move in a domain with multiple well-spaced polygonal obstacles. For such obstacles, the distance between any pair of non-adjacent obstacle edges is at least ε.

6.3.1 Lower Bound

▶ **Lemma 6.6.** *The total number of critical events for a set of n entities, each moving amidst a set of well-spaced obstacles $\mathcal{O}$ along a piecewise linear trajectory with τ vertices, is $\Omega(\tau(n^2 + nm))$, where m is the total complexity of $\mathcal{O}$.*

Proof. We describe a construction in which the entities move along lines that yields $\Omega(nm)$ critical events. We repeat this construction in $\Omega(\tau)$ time intervals.

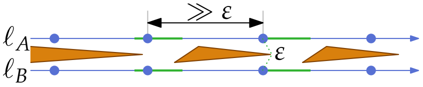


Figure 6.3: The lower bound construction for well-spaced obstacles. The entities of a pair a, b are within distance ε from each other when both move in a green interval.

Since we already have a $\Omega(\tau n^2)$ lower bound for entities moving in $\mathbb{R}^2$ without obstacles, the lemma then follows.

The construction that we use is sketched in Figure 6.3. We have two horizontal lines ℓ_A and ℓ_B that are within vertical distance ε of each other. Our obstacles essentially form a wall separating the two lines that has $\Theta(m)$ openings. Each obstacle is triangular, and thus well-spaced by itself. Furthermore, the obstacles are at distance at least ε from each other, so $\mho$ is well-spaced. Our set of entities consists of two equal-sized subsets A and B. The entities move in pairs; one entity a from A and one entity b from B. Throughout the movement they maintain $a_x = b_x$, and stay far away from any other entities. It is easy to see that this yields $\Omega(nm)$ critical events as claimed. ☐

6.3.2 Upper Bound

The obstacles are well-spaced, so if two entities are at geodesic distance ε the geodesic consists of at most three line segments. We now start with some bounds on the total number of ε-events.

▶ **Observation 6.7.** *There are at most $O(\tau n^2)$ ε-events where the geodesic between the two entities involved is a single line segment.*

▶ **Lemma 6.8.** *Let $\mathcal{E}$ be a set of n entities, each moving amidst a set of well-spaced obstacles $\mho$ along a piecewise linear trajectory with τ vertices. The number of ε-events is at most $O(\tau n^2 m)$, where m is the total complexity of $\mho$.*

Proof. By Observation 6.7 there are $O(\tau n^2)$ ε-events in which the geodesic is a single line segment. We now bound the number of ε-events for which the geodesic contains an obstacle vertex v by $O(\tau n^2)$. The lemma then follows. Fix two entities a and b. Each trajectory edge intersects the ε-disk centered at v at most once. Hence, there are $O(\tau)$ time intervals during which both a and b move along a line, and are within distance ε from v. Clearly, all ε-events involving a and b occur within one of these intervals. Since the obstacles are well-spaced, the ε-disk contains at most three edges: the two edges connected to v and at most one edge adjacent to both these edges. It follows that the function ς_{ab} consists of at most $O(1)$ pieces during such an interval. Hence, there can be at most a constant number of ε-events per interval. ☐

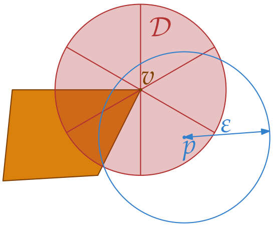


Figure 6.4: The ε-disk $\mathcal{D}$ (red) centered at v subdivided into six wedges. The distance between any pair of points p and q in the same wedge is at most ε.

Next, we show that the number of critical events can be only $O(\tau(n^2 + m\lambda_4(n)))$. Clearly, the number of critical events at which the geodesic is a single line segment is also at most $O(\tau n^2)$ (Observation 6.7). We now bound the number of critical events where two sets of entities become ε-connected or ε-disconnected, and the geodesic between them consists of two line segments, connected via an obstacle vertex v.

Let $\mathcal{D}$ be the disk of radius ε centered at v, and consider a subdivision of $\mathcal{D}$ into six equal-size sectors or *wedges*. See Figure 6.4. We make sure that the obstacle containing v intersects at least two wedges. Let W be a wedge. For any pair of points p and q in W, the Euclidean distance between p and q is at most ε. Let $\mathcal{E}_W(t) \subseteq \mathcal{E}$ denote the set of entities that lie in W at time t.

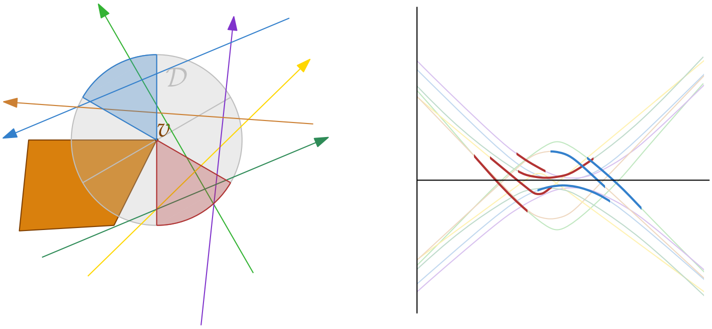
▶ **Observation 6.9.** *At any time t, there is at most one maximal set of ε-connected entities G that has entities in wedge W, that is, for which $G \cap \mathcal{E}_W(t) \neq \emptyset$.*

▶ **Corollary 6.10.** *At any time t, there is at most one maximal set of ε-connected entities G such that $\mathcal{E}_W(t) \subseteq G$.*

When two maximal sets of ε-connected entities $\mathcal{E}_R$ and $\mathcal{E}_B$ become ε-connected or ε-disconnected at time t via vertex v, then the entities $r \in \mathcal{E}_R$ and $b \in \mathcal{E}_B$ that form their closest pair must both lie in $\mathcal{D}$ at time t. More specifically, since the geodesic between r and b uses vertex v, r and b must lie in different wedges. Let R and B denote the wedges that contain r and b, respectively. We now show that the total number of critical events involving entities in wedges R and B is $O(\tau\lambda_4(n))$. By Corollary 6.10 it then follows that each such event corresponds to exactly one pair of ε-connected sets. Since there are only $\binom{6}{2} = 15$ pairs of wedges, there are also at most $O(\tau\lambda_4(n))$ times when two maximal sets of ε-connected entities are at distance exactly ε and are connected via vertex v.

▶ **Lemma 6.11.** *The total number of critical events involving entities in wedges R and B is $O(\tau\lambda_4(n))$.*

Proof. Given an entity $a \in \mathcal{E}$ we define two partial functions ϱ_a and β_a as



(a)

(b)

Figure 6.5: (a) A set of entities. (b) The corresponding sets of partial functions $\mathscr{R}$ (red) and $\mathscr{B}$ (blue). Critical events correspond to intersections between the lower envelope of $\mathscr{R}$ and the upper envelope of $\mathscr{B}$.

follows:

$$\varrho_a(t) = \begin{cases} \xi_{av}(t) - \varepsilon/2 & \text{if } a \in \mathscr{E}_R(t) \\ \perp & \text{otherwise,} \end{cases} \qquad \beta_a(t) = \begin{cases} -\xi_{av}(t) + \varepsilon/2 & \text{if } a \in \mathscr{E}_B(t) \\ \perp & \text{otherwise,} \end{cases}$$

where $\perp$ denotes undefined. Furthermore, let $\mathscr{R} = \{\varrho_r \mid r \in \mathscr{E}\}$ and $\mathscr{B} = \{\beta_b \mid b \in \mathscr{E}\}$. See Figure 6.5. It now follows that for any two entities $r \in \mathscr{E}_R(t)$ and $b \in \mathscr{E}_B(t)$ the length of the path from r via v to b is ε if and only if $\varrho_r(t) = \beta_b(t)$. Thus, the number of times entities in R become ε-connected or ε-disconnected via vertex v is at most the number of intersection points between the lower envelope of $\mathscr{R}$ and the upper envelope of $\mathscr{B}$. Next, we show that this number of intersection points is at most $O(\tau\lambda_4(n))$.

Each trajectory consists of $\tau - 1$ edges, each of which intersects wedge R in a single line segment. Hence, for each entity a, the function ϱ_a is defined on at most $\tau - 1$ maximal contiguous intervals $I_a^1, .., I_a^{\tau-1}$. Thus, by Lemma 6.1 the lower envelope $\mathcal{L}$ of $\mathscr{R}$ has complexity at most $O(\tau\lambda_4(n))$. Similarly, the upper envelope $\mathcal{U}$ of $\mathscr{B}$ has complexity $O(\tau\lambda_4(n))$. It follows that there are also $O(\tau\lambda_4(n))$ time intervals such that both $\mathcal{L}$ and $\mathcal{U}$ are represented by a simple hyperbolic function. In each such interval $\mathcal{L}$ and $\mathcal{U}$ intersect each other at most twice. Hence, the total number of intersection points is $O(\tau\lambda_4(n))$. ☐

It now follows that the total number of critical events at which the geodesic contains an obstacle vertex is $O(m\tau\lambda_4(n))$. We conclude:

▶ **Theorem 6.12.** *Let $\mathscr{E}$ be a set of n entities, each moving amidst a set of well-spaced obstacles $\mathscr{O}$ along a piecewise linear trajectory with τ vertices. The number of critical events is at most $O(\tau(n^2 + m\lambda_4(n)))$, where m is the total complexity of $\mathscr{O}$.*

6.3.3 Algorithm

We now show how to compute the Reeb graph $\mathcal{R}$ in case the entities move amidst well-spaced obstacles. At first glance, it seems that we can compute all critical events using the same approach as used in the upper bound proof. Indeed, this allows us to find all times at which critical events occur. However, to construct the Reeb graph we also need to know the sets of entities involved at each critical event, e.g. we want to know that a set $\mathscr{E}'$ splits into subsets R and B. Unfortunately, there does not seem to be an efficient, i.e. sub-linear, way to obtain this information, nor can we easily maintain the ε-connected sets of entities without considering all ε-events. It is easy to compute all ε-events in $O(\tau n^2 m)$ time, using the approach described in Lemma 6.8. Once we have computed all ε-events, we can construct the Reeb graph using the same method as before (Theorem 5.7). This takes $O(\log n)$ time per ε-event. Thus, we conclude:

▶ **Theorem 6.13.** *Let $\mathscr{E}$ be a set of n entities, each moving amidst a set of well-spaced obstacles $\mho$ along a piecewise linear trajectory with τ vertices. The Reeb graph $\mathcal{R}$ representing the movement of the entities in $\mathscr{E}$ has size $O(\tau(n^2 + m\lambda_4(n)))$ and can be computed in $O(\tau n^2 m \log n)$ time, where m is the total complexity of $\mho$.*

6.4 General Obstacles

Finally, we study the most general case in which the entities move amidst multiple obstacles, and there are no restrictions on the locations, shape, or size of the obstacles.

6.4.1 Lower Bound

▶ **Lemma 6.14.** *The total number of critical events for a set of n entities, each moving amidst a set of obstacles $\mho$ along a piecewise linear trajectory with τ vertices, is $\Omega(\tau(n^2 + nm\min\{n, m\}))$, where m is the total complexity of $\mho$.*

Proof. We describe a construction in which the entities move along lines that yields $\Omega(nmk)$ critical events, with $k = \min\{n, m\}$. We again repeat this construction $\Omega(\tau)$ times.

The basic idea is to create $\Omega(k)$ stationary entities, $\Omega(n)$ moving entities, and $\Omega(m)$ "entrances" from which a moving entity can become connected with a stationary entity. Each stationary entity is surrounded by an obstacle. The distance from such a stationary entity s to an entrance leading to s, will be approximately ε. So an entity gets ε-connected with s only if it is directly in front of the entrance. We make sure that each stationary entity is reachable

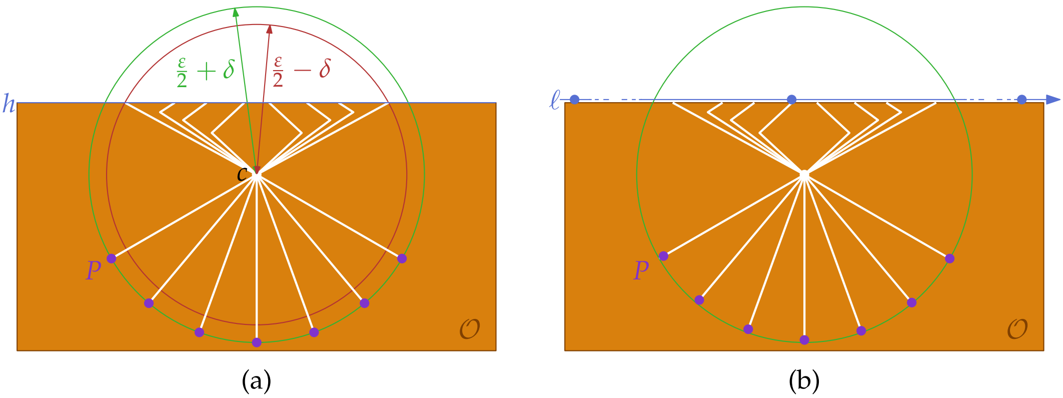

h
$\frac{\varepsilon}{2}+\delta$
$\frac{\varepsilon}{2}-\delta$
c
P
$\mathcal{O}$
(a)
ℓ
P
$\mathcal{O}$
(b)

Figure 6.6: The lower bound construction for general obstacles. (a) Constructing the passages through obstacle $\mathcal{O}$. (b) The final construction.

from all entrances. Hence, each time that one of the $\Omega(n)$ moving entities passes an entrance it will generate $\Omega(k)$ critical events. Since all $\Omega(n)$ moving entities encounter all $\Omega(m)$ entrances we get at least $\Omega(nmk)$ critical events.

Let c be a point in the plane, let $\delta > 0$ be a small value, and let P be a set of $\Omega(k)$ points on the lower half of the circle with center c and radius $\varepsilon/2 + \delta$. We place a large rectangular obstacle $\mathcal{O}$ containing c and all points in P such that the (shortest) distance from c to the top side h of $\mathcal{O}$ is smaller than $\varepsilon/2 - \delta$.

We now carve $\Omega(k + m)$ passages through $\mathcal{O}$. The first $k' = \Omega(k)$ connect c to each point in P. The remaining $m' = \Omega(m)$ connect c to the top side h of the obstacle $\mathcal{O}$. The first k' passages all have length exactly $\varepsilon/2 + \delta$, and we make sure that the remaining m' passages all have length exactly $\varepsilon/2 - \delta$. We can do this with at most one bend in each passage. See Figure 6.6(a). The distance from any point in P to the top side h of $\mathcal{O}$, via any of the m' passages, is now ε, and the distance between any two points in P is strictly larger than ε.

We place a stationary entity on each point in P, and we let $\Omega(n)$ entities move from left to right on a horizontal line ℓ containing h (we can move ℓ upwards a bit later to make sure the entities do not intersect the obstacle). We make sure that at any time the distance between two of these moving entities is larger than ε, so they are never in the same ε-connected set. When an entity e arrives at an entrance, that is, an opening of one of the top passages, it is at distance ε to the points in P. Hence, we have a critical event where e connects with all entities at points in P. We can make sure that e generates an event with (the entity on) *each* point in P by moving each point in P by a small unique amount towards c. Figure 6.6(b) shows the resulting construction. □

6.4.2 Upper Bound

We again start by bounding the total number of ε-events.

▶ **Lemma 6.15.** *Let $\mathcal{E}$ be a set of n entities, each moving amidst a set of obstacles $\mathcal{O}$ along a piecewise linear trajectory with τ vertices. The number of ε-events is at most $O(\tau n^2 m^2)$, where m is the total complexity of $\mathcal{O}$.*

Proof. As before, there are $O(\tau n^2)$ ε-events for which the geodesic consists of a single line segment. We proceed to bound the number of ε-events for which the geodesic contains an obstacle vertex v.

Consider the shortest path map Ψ with starting point v: a planar subdivision such that all points in a region of Ψ have the same combinatorial geodesic to v [68]. This subdivision has complexity $O(m)$, and its edges are either line segments or hyperbolic arcs. It follows that any line intersects at most $O(m)$ edges of Ψ, and thus visits at most $O(m)$ regions of Ψ. Therefore, each entity a visits at most $O(\tau m)$ regions. The geodesic distance between a, moving along a line segment in a region R of Ψ, and v is a simple hyperbolic function.

Fix a pair of entities a and b. There are $O(\tau m)$ time intervals such that a moves along a line in region R_a of Ψ and b moves along a line in region R_b. During such an interval the distance between a and b via v is a simple function. Hence, there can be at most two times in each interval such that $\varsigma_{av}(t) + \varsigma_{vb}(t) = \varepsilon$. Thus, the total number of times such that the distance between a and b via an obstacle vertex is exactly ε is $O(\tau m^2)$. It follows that the number of ε-events involving a and b is then also at most $O(\tau m^2)$. The lemma follows. ☐

As in the case of well-spaced obstacles not all of these ε-events can be critical events. We now fix an obstacle vertex v, and show that there are at most $O(\tau m^2 \lambda_4(n))$ critical events involving v. To this end, we again decompose the (geodesic) ε-disk centered at v into regions such that each region corresponds to at most one maximal set of ε-connected entities. Each critical event involving v also involves two maximal ε-connected sets, and thus two regions in this decomposition. We show that we have to consider only $O(m)$ pairs of such regions, and that for each pair there can be at most $O(\tau m \lambda_4(n))$ critical events. Since we have $O(m)$ obstacle vertices this gives us a total bound of $O(\tau m^3 \lambda_4(n))$. Finally, we show that by counting only the critical events where v is the *first* vertex on the geodesic we can reduce this number to $O(\tau m^2 \lambda_4(n))$, which is close to optimal if n and m are in the same order of magnitude.

Let $\mathcal{D}_\varepsilon$ denote the geodesic ε-disk centered at v, and let $\mathcal{D}_{\varepsilon/2}$ denote the geodesic $(\varepsilon/2)$-disk centered at v. Clearly, the geodesic distance between any two points in $\mathcal{D}_{\varepsilon/2}$ is at most ε, thus we observe:

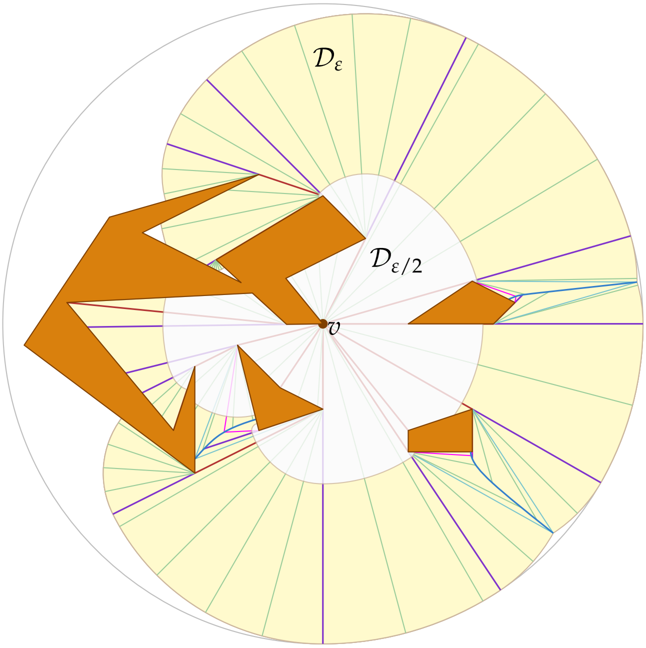


Figure 6.7: Subdivision Φ. The color of the edge indicates its type: the red edges originate from shortest paths, the purple and blue edges from the shortest path map, the cyan edges from the subdivision in "triangular sectors", the light green edges guarantee that the maximum angle at the routing point is at most $\pi/12$, and the pink edges guarantee monotonicity.

▶ **Observation 6.16.** *At any time t there is at most one maximal ε-connected set of entities G such that $G_{\mathcal{D}_{\varepsilon/2}}(t) \neq \emptyset$, and thus $\mathcal{E}_{\mathcal{D}_{\varepsilon/2}}(t) \subseteq G$.*

Let $\mathcal{A} = \mathcal{D}_\varepsilon \setminus \mathcal{D}_{\varepsilon/2}$. We decompose $\mathcal{A}$ into $O(m)$ regions such that for each region R we have that

(i) the geodesic distance between two points $p, q \in R$ is at most ε,
(ii) any two points $p, q \in R$ have the same (combinatorial) geodesic to v, and
(iii) the boundary of R has constant complexity.

Let Φ denote this decomposition of $\mathcal{A}$. It follows that at any time, each region R in Φ contains entities from at most one maximal ε-connected set G. That is, $\mathcal{E}_R(t) \subseteq G$. It is now easy to see that any critical event involving v involves the maximal set of ε-connected entities $G_{\varepsilon/2}$ corresponding to $\mathcal{D}_{\varepsilon/2}$, and a maximal set of ε-connected entities G_R corresponding to a region R of Φ. Hence, there are only $O(m)$ pairs of regions that can be associated with a critical event involving v. We now show how to construct Φ, and how to bound the number of events corresponding to a single pair of regions.

Obtaining subdivision Φ. Let Φ' be the overlay of the shortest path map with root v (restricted to $\mathcal{D}_\varepsilon$), and all shortest paths from v to obstacle vertices in $\mathcal{D}_\varepsilon$.

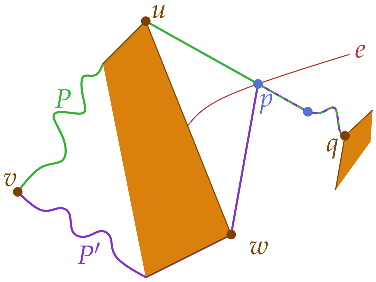


Figure 6.8: If P is a shortest path from q to v it cannot intersect an edge e of the shortest path map.

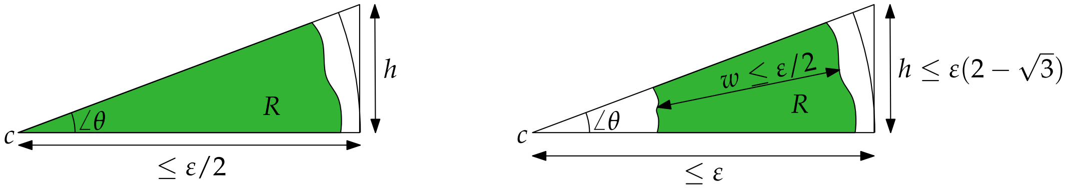
▶ **Lemma 6.17.** *Φ' has complexity $O(m)$.*

Proof. The shortest path map $\mathcal{M}$ with root v has complexity $O(m)$, and the shortest paths to obstacle vertices form a rooted tree T (with root v) of size $O(m)$ [68]. We now show that the edges of T and $\mathcal{M}$ intersect each other only at obstacle vertices. It then follows that the complexity of their overlay is also $O(m)$.

Consider the shortest path P from obstacle vertex q to v, and assume by contradiction that this path intersects an edge e of $\mathcal{M}$ in point p. It follows that e is a (weighted) bisector for the obstacle vertices u and w, that is, for any point on e —so in particular for point p— there are two shortest paths to v, one via u and one via w. See Figure 6.8. Assume without loss of generality that P is the path via u, and let P' be the other path via w. Path P' is a shortest path with a vertex at p. It is easy to show that any shortest path contains only obstacle vertices (except for its endpoints), and that any path with a non-obstacle vertex (i.e. a vertex where the path just bends) cannot be a shortest path. It follows that P' is not a shortest path. Since P has the same length as P' it cannot be a shortest path either. Contradiction. ☐

The edges of Φ' are either line segments or hyperbolic arcs [68]. Since Φ' is a refinement of the shortest path map, all points in a region R in Φ' have the same geodesic g to v (except for the starting edge). Hence, each region R is star-shaped, and has a vertex c that lies inside its kernel. This vertex c is the second vertex on each geodesic g. We refer to c as the *routing point* of R.

Next, we further subdivide each region R in Φ'. We add edges $\overline{cu}$ between the routing point c and all boundary vertices u of R. Each region is now bounded by two line segments $\overline{cu}$ and $\overline{cw}$ and a segment $\widetilde{cw}$. This segment $\widetilde{cw}$ is either a line segment, or a hyperbolic arc. We further add edges $\overline{cz}$ between c and points z on $\widetilde{cw}$ such that the angle at c is at most $\theta = \pi/12$. In case $\widetilde{cw}$ is a hyperbolic arc we make sure that the hyperbolic function describing this arc is monotonic. To this end, we add at most one additional edge $\overline{cz}$ to the point



(a) (b)

Figure 6.9: (a) When $c \in \mathcal{A}$, region R is contained in a sector with radius $\varepsilon/2$ and angle $\theta = \pi/12$ at c. (b) When c does not lie in $\mathcal{A}$, the sector containing R has height h.

z on $\widetilde{cw}$ with maximum curvature. All these new edges are contained in R and do not intersect each other. It follows that the total complexity, summed over all regions in the subdivision, is still $O(m)$. Let Φ denote the resulting subdivision, restricted to $\mathcal{A}$. See Figure 6.7.

▶ **Lemma 6.18.** *Let R be a region in Φ. For any two points $p, q \in R$ the Euclidean distance $\|pq\|$ between p and q is at most $\varepsilon\sqrt{29/4 - 4\sqrt{3}}$.*

Proof. Let c denote the routing point of R. We distinguish two cases, depending on whether or not c lies in $\mathcal{A}$.

Consider the case $c \in \mathcal{A}$. Since c is the second vertex on the shortest path from any point p in R to v, and $c \notin D_{\varepsilon/2}$, the distance from p to c is at most $\varepsilon/2$. It follows that R is contained in a circular sector with its apex at c, radius $\varepsilon/2$, and angle $\theta = \pi/12$. Simple trigonometry shows that the Euclidean distance between any pair of points in such a sector, and thus in R, is at most $\varepsilon/2$.

Consider the case $c \notin \mathcal{A}$. Similar to the previous case, we get that R is contained in a circular sector with its apex at c, radius ε, and angle $\pi/12$. The *height* h of this sector (see Figure 6.9) is at most $\varepsilon\tan(\pi/12) \leq \varepsilon(2-\sqrt{3})$. To show that the distance between a pair of points p and q in R is at most ε we rotate R and the sector containing R around c until $\overline{cp}$ is horizontal. Since $R \subseteq \mathcal{A}$ it follows that the x-component of $\|pq\|$ is at most $w \leq \varepsilon/2$. The y-component is bounded by h. Thus, we have $\|pq\| \leq \sqrt{w^2 + h^2} \leq \varepsilon\sqrt{29/4 + 4\sqrt{3}}$. This completes the proof. □

▶ **Lemma 6.19.** *Let R be a region in Φ. For any two points $p, q \in R$ the geodesic distance $\varsigma(p,q) = \varsigma(g(p,q))$ between p and q is at most ε.*

Proof. Let p and q be a pair of points in R. If $g(p,q) = \overline{pq}$ then we have $\varsigma(p,q) = \|pq\| \leq \varepsilon\sqrt{29/4 - 4\sqrt{3}} \leq \varepsilon$ by Lemma 6.18. Otherwise, $\overline{pq}$ intersects an obstacle, and thus a concave part of the boundary of R. If we follow the boundary of R we encounter, in order, a line segment $\overline{au}$, a segment $\widetilde{uw}$, a line segment $\overline{wb}$, and possibly a part $\widetilde{ab}$ of the boundary of $D_{\varepsilon/2}$ connecting b to a. If $\widetilde{ab}$ is empty then $a = b = c$, where c is the routing point of R. The routing point c is the second vertex on all shortest paths from points in R to v. Hence, the area bounded by $\overline{ca}$, $\widetilde{ab}$, and $\overline{bc}$ is empty of obstacles. It follows that $g(p,q)$ intersects $\widetilde{uw}$.

By construction, segment $\widetilde{uw}$ is described by a monotonic hyperbolic function. Since this function is monotonic, it follows that the dilation of $\widetilde{uw}$ is at most $\sqrt{2}$. That is, for any pair of points r and s on $\widetilde{uw}$, the length of the curve $\widetilde{rs}$ is at most $\|rs\|\sqrt{2}$. Combining this with the result from Lemma 6.18 it follows that $g(p,q)$ has length at most $\varepsilon\sqrt{29/4 - 4\sqrt{3}}\sqrt{2} \leq \varepsilon$. ☐

▶ **Lemma 6.20.** *Subdivision Φ has complexity $O(m)$ and each region $R \in \Phi$ has the following properties:*

(i) *the geodesic distance between two points $p, q \in R$ is at most ε,*
(ii) *any two points $p, q \in R$ have the same geodesic to v (excluding the starting edge), and*
(iii) *the boundary of R has constant complexity.*

Proof. Property (i) follows directly from Lemma 6.19, and Property (ii) follows from the fact that Φ is a refinement of the shortest path map. Each region is bounded by three or four segments, depending if the routing point c lies in $\mathcal{A}$ or not. If $c \in \mathcal{A}$, region R is bounded by three segments. Otherwise, R is bounded by three segments and a part of $D_{\varepsilon/2}$. However, as all shortest paths from points in R to v use point c, it follows that this part of $D_{\varepsilon/2}$ is also a single hyperbolic segment. This proves Property (iii). ☐

Bounding the number of critical events for a pair of regions. Next, we fix a region R in Φ, and show that the number of critical events involving v, R, and $D_{\varepsilon/2}$, is at most $O(\tau\lambda_4(n))$.

▶ **Lemma 6.21.** *Let R be any region of Φ, and let G_R be the maximal set of ε-connected entities corresponding to R. The (geodesic) distance between G_R and v is given by a piecewise hyperbolic function with $O(\tau\lambda_4(n))$ pieces.*

Proof. The boundary of R has constant complexity, so each entity in G_R intersects region R in $O(\tau)$ time-intervals. Furthermore, all points in R have the same combinatorial geodesic, so during any such an interval, the distance to v is given by a simple hyperbolic function. Thus, the distance function between G_R and v corresponds to the lower envelope of a set of hyperbolic functions. Lemma 6.1 now completes the proof. ☐

Fix a region R, let

$$\beta_a(t) = \begin{cases} -\varsigma(a(t), v) + \varepsilon & \text{if } a(t) \in R \\ \bot & \text{otherwise,} \end{cases}$$

and let $\mathcal{U}$ be the upper envelope of $\{\beta_a(t) \mid a \in \mathscr{E}\}$. It follows from Lemma 6.21 that $\mathcal{U}$ has complexity $O(\tau\lambda_4(n))$.

Now consider the entities in the inner region $\mathcal{D}_{\varepsilon/2}$. The function ς_{av} expressing the geodesic distance between a and v is piecewise hyperbolic and consists of $O(m\tau)$ pieces. Let $\mathcal{L}$ denote the lower envelope of all functions ϱ_a, $a \in \mathscr{E}$, where $\varrho_a(t) = \varsigma_{av}(t)$ if $\varsigma_{av}(t) \leq \varepsilon/2$ and $\bot$ otherwise. It follows from Lemma 6.1 that $\mathcal{L}$ has complexity $O(m\tau\lambda_4(n))$.

As with the well-spaced obstacles, all critical events in which the entities involved lie in $\mathcal{D}_{\varepsilon/2}$ and R at the time of the event correspond to intersections of $\mathcal{L}$ and $\mathcal{U}$. To bound the number of intersections, and thus the number of critical events, we now (again) partition the domain of $\mathcal{L}$ and $\mathcal{U}$ (i.e., time) into sets $D_1, .., D_k$ such that in each D_i the lower envelope $\mathcal{L}$ and the upper envelope $\mathcal{U}$ intersect at most twice. It is easy to partition the domain into $k = O(|\mathcal{L}| + |\mathcal{U}|) = O(\tau\lambda_4(n) + m\tau\lambda_4(n)) = O(m\tau\lambda_4(n))$ intervals with this property. Hence, we get $O(m\tau\lambda_4(n))$ critical events involving vertex v and the pair of regions $(R, \mathcal{D}_{\varepsilon/2})$. This gives a total of $O(m^3\tau\lambda_4(n))$ critical events.

Counting critical events where v is the first vertex on the geodesic. The above argument over-counts the number of critical events; a critical event is counted for all vertices on the geodesic. To avoid this, we charge a critical event to an obstacle vertex v only if it is the *first* vertex on the geodesic (now each critical event is counted at most twice). In terms of the intersections between $\mathcal{L}$ and $\mathcal{U}$ this means that we wish to count the intersection only if it lies on a piece of $\mathcal{L}$ corresponding to the Euclidean distance between $G_{\varepsilon/2}$ and v.

We will use the same global approach as before. However, we use a different partition of the domain of $\mathcal{L}$ and $\mathcal{U}$. We will show that we can partition the domain into $O(\tau\lambda_4(n))$ sets, such that in each set there are at most two times, t_1 and t_2, such that $\mathcal{L}$ intersects $\mathcal{U}$, and $\mathcal{L}(t_i)$ is the Euclidean distance between $G_{\varepsilon/2}$ and v. Thus, the number of critical events involving the pair of regions $\mathcal{D}_{\varepsilon/2}$ and R, and such that v is the first vertex on the geodesic is $O(\tau\lambda_4(n))$. Since we have $O(m)$ pairs of regions and m obstacle vertices this gives a total of $O(m^2\tau\lambda_4(n))$ critical events as desired.

Let a be an entity, and let ξ^i_{av} be the i^{th} piece of function ξ_{av} expressing the Euclidean distance between a and v. We define

$$\mathcal{L}^i_a(t) = \begin{cases} \mathcal{L}(t) & \text{if } \mathcal{L}(t) = \varrho_a(t) = \xi^i_{av}(t) \\ \bot & \text{otherwise.} \end{cases}$$

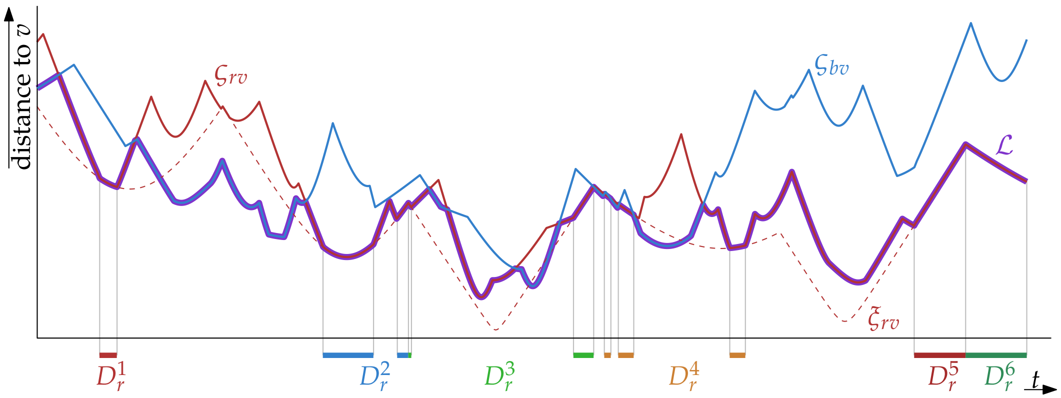


Figure 6.10: The lower envelope $\mathcal{L}$ (purple) of two geodesic distance functions ϱ_r (red) and ϱ_b (blue). We have $\mathcal{L}^i_r(t) = \mathcal{L}(t)$ if $\mathcal{L}(t)$ lies on the i^{th} piece of the Euclidean distance function ξ_{rv}. These functions $\mathcal{L}^i_r$ consist of multiple disjoint pieces, as indicated by the colored time intervals.

The function $\mathcal{L}^i_a$ is defined at time t if and only if the lower envelope $\mathcal{L}$ uses the i^{th} piece of ξ^i_{av}. That is, when the distance between v and $G_{\varepsilon/2}$ is the Euclidean distance between v and entity a, and a is traveling along the i^{th} edge of its trajectory. Note that $\mathcal{L}^i_a$ may consist of several disjoint pieces of $\mathcal{L}$. See Figure 6.10. We are now interested in the number of intersection points between all functions $\mathcal{L}^i_a$ and $\mathcal{U}$.

Any piece ξ^i_{av} can intersect a single piece of $\mathcal{U}$ at most twice. Since for any $t \in \mathrm{dom}(\mathcal{L}^i_a)$, $\mathcal{L}^i_a(t) = \xi^i_{av}(t)$, the same applies for $\mathcal{L}^i_a$. So even though $\mathcal{L}^i_a$ can consist of many pieces, together they intersect a single piece of $\mathcal{U}$ at most twice.

We have $O(\tau n)$ functions $\mathcal{L}^i_a$. We split the domain $D^i_a = \mathrm{dom}(\mathcal{L}^i_a)$ into two sets if $\mathcal{U}$ has a break point. Let $D_1, .., D_k$ denote the resulting sets. Since every break point of $\mathcal{U}$ creates one additional set, we have that $k = O(\tau n + \tau\lambda_4(n)) = O(\tau\lambda_4(n))$. In each of these sets $\mathcal{L}$ and $\mathcal{U}$ intersect at most twice, and together they include all intersection points where $\mathcal{L}$ correspond to the Euclidean distance between v and $G_{\varepsilon/2}$. Hence, the total number of intersections, and therefore also the number of critical events, is at most $O(\tau\lambda_4(n))$. We thus obtain the following result:

▶ **Theorem 6.22.** *Let $\mathcal{E}$ be a set of n entities, each moving amidst a set of obstacles $\mathcal{O}$ along a piecewise linear trajectory with τ vertices. The number of critical events is at most $O(\tau(n^2 + m^2\lambda_4(n)))$, where m is the total complexity of $\mathcal{O}$.*

6.4.3 Algorithm

We again explicitly compute all ε-events in order to construct the Reeb graph $\mathcal{R}$. We follow the approach from Lemma 6.15. That is, we compute the shortest path map Ψ with root v, and for each pair of entities a and b we trace their trajectories through Ψ. For each of the $O(\tau m)$ pairs of regions visited, we construct ς_{ab} and find the ε-events. Computing the shortest path map with root v takes $O(m \log m)$ time [68]. Tracing the trajectories and computing the distance functions takes time proportional to the number of regions visited. Hence, we spend $O(\tau m)$ time for each pair. It follows that the total time required to compute all ε-events is $O(m(m \log m + n^2 \tau m)) = O(\tau n^2 m^2 + m^2 \log m)$. Computing $\mathcal{R}$ again takes $O(\log n)$ time per ε-event. We obtain the following result.

▶ **Theorem 6.23.** *Let $\mathcal{E}$ be a set of n entities, each moving amidst a set of obstacles $\mathcal{O}$ along a piecewise linear trajectory with τ vertices. The Reeb graph $\mathcal{R}$ representing the movement of the entities in $\mathcal{E}$ has size $O(\tau(n^2 + m^2 \lambda_4(n)))$ and can be computed in $O(\tau n^2 m^2 \log n + m^2 \log m)$ time, where m is the total complexity of $\mathcal{O}$.*

6.5 Concluding Remarks

We studied the trajectory grouping structure for entities moving amidst obstacles. To this end, we analyzed the number of times when two sets of entities are at distance ε from each other. Our results for various types of obstacles can be found in Table 6.1. These bounds on the number of critical events also give a bound on the size of the Reeb graph $\mathcal{R}$. This in turn gives bounds on the number of maximal groups: if the Reeb graph has size $O(|\mathcal{R}|)$ there are $O(|\mathcal{R}|n)$ maximal groups. Furthermore, we presented efficient algorithms to compute $\mathcal{R}$, which leads to efficient algorithms to compute the grouping structure.

The most intriguing open question is whether the Reeb graph can be constructed using only the critical events, that is, in an output-sensitive manner. The difficulty with the approach that we use appears to be that one would need a dynamic data structure for maintaining a subdivision of a set (the groups), that supports efficient split and merge operations. Thus, there may be fundamental graph-theoretical obstacles to this approach. However, it is not clear that this is the only possible approach to compute $\mathcal{R}$.

An interesting direction of future work is to extend the grouping structure for entities moving in more realistic environments, for instance modeled by weighted regions. This starts with interesting modeling questions since distances are related to the speed of the entities. For example: should the distance for two entities, say sheep, to be directly connected be larger on a muddy field than it is on a concrete courtyard, or do the sheep need to be closer together in the field to be considered a group?

Although we developed the technical machinery presented here with the goal of extending the trajectory grouping structure, we foresee wider applications for our techniques. We believe our work will serve as a starting point for more general research related to moving entities and geodesic distances. For example, we can consider trajectory similarity measures in the presence of obstacles.

Chapter 7

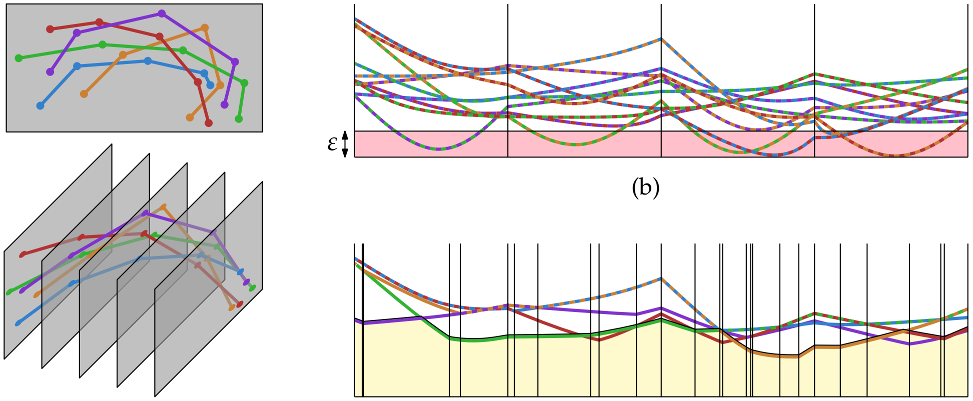
Central Trajectories

In this chapter we study the problem of computing a suitable representative for a set of similar trajectories. The representative should capture the defining features of all trajectories in the input set. We introduce a *time-dependent* representative, a representative that includes the spatial as well as the temporal component of the trajectories, and present efficient algorithms to compute such a representative. Our representative, a *central trajectory* $\mathfrak{C}$, consists of pieces of the input trajectories and switches from one entity to another only if they are within a small distance of each other. Furthermore, at any time t, the point $\mathfrak{C}(t)$ is as central as possible. We measure centrality in terms of the radius of the smallest disk centered at $\mathfrak{C}(t)$ enclosing all entities at time t, but we discuss how our techniques can be adapted to other measures of centrality.

Ideally, we would output a trajectory $\mathfrak{C}$ such that at any time t, $\mathfrak{C}(t)$ is the point (entity) that is closest to its farthest entity. Unfortunately, when the entities move in $\mathbb{R}^d$ for $d > 1$, this may cause discontinuities. Such discontinuities are unavoidable: if we insist that the output trajectory consists of pieces of input trajectories *and* is continuous, then in general, there will be no opportunities to switch from one trajectory to another. Thus, we are effectively choosing one of the input trajectories. This is undesirable, as no individual trajectory may be a good representative. At the same time, we do not want to output a trajectory with arbitrarily large discontinuities. An acceptable compromise is to allow discontinuities, or *jumps*, but only over small distances, controlled by a parameter ε.

Problem description. We are given a set $\mathcal{E}$ of n entities, each moving along a piecewise-linear trajectory in $\mathbb{R}^d$ consisting of τ edges. We assume that all trajectories have their vertices at the same times, i.e. times $t_0, .., t_\tau$.

Consider a pair of entities $a, b \in \mathcal{E}$, and recall that $\xi_{ab}(t) = \|a(t)b(t)\|$ is the distance between a and b at time t. We denote the distance from a to the entity farthest away from a by $\Xi_a(t) = \Xi(a, t) = \max_{b \in \mathcal{E}} \xi_{ab}(t)$. Figure 7.1 shows five example trajectories, and illustrates the pairwise distances and resulting Ξ functions for these trajectories. For ease of exposition, we assume that the



(b)

(a)

(c)

Figure 7.1: (a) Two views of five moving entities and their trajectories. (b) The pairwise distances between the entities as a function over time. (c) The functions Ξ_a, and in yellow the area representing $\Psi(\mathfrak{C})$, the quantity that we wish to minimize.

trajectories are in general position: that is, no three trajectories intersect in the same point, and no two pairs of entities are at distance ε from each other at the same time.

A *trajectoid* is a function that maps time to the set of entities $\mathscr{E}$, with the restriction that at discontinuities the distance between the entities involved is at most ε. Intuitively, a trajectoid corresponds to a concatenation of pieces of the input trajectories in such a way that two consecutive pieces match up in time, and the end point of the former piece is within distance ε from the start point of the latter piece. In Figure 7.1(b), a trajectoid may switch between a pair of entities when their pairwise distance function lies in the bottom strip of height ε. More formally, for a trajectoid $\mathfrak{T}$ we have that

- at any time t, $\mathfrak{T}(t) = a$ for some $a \in \mathscr{E}$, and
- at every time t where $\mathfrak{T}$ has a discontinuity, that is, $\mathfrak{T}$ *jumps* from entity a to entity b, we have that $\xi_{ab}(t) \leq \varepsilon$.

Note that this definition still allows for a series of jumps within an arbitrarily short time interval $[t, t + \delta]$, essentially simulating a jump over distances larger than ε. To make the formulation cleaner, we slightly weaken the second condition, and allow a trajectoid to have discontinuities with a distance larger than ε, provided that such a large jump can be realized by a sequence of small jumps, each of distance at most ε. When it is clear from the context, we will write $\mathfrak{T}(t)$ instead of $\mathfrak{T}(t)(t)$ to mean the location of entity $\mathfrak{T}(t)$ at time t. We now wish to

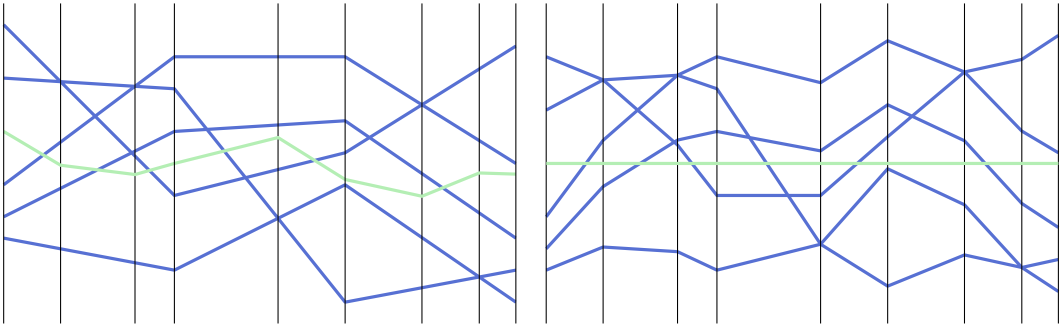

(a)
(b)

Figure 7.2: (a) A set of trajectories and the ideal trajectory $\mathcal{I}$. The breakpoints in the ideal trajectory partition time into $O(n\tau)$ intervals. (b) The trajectories after transforming $\mathcal{I}$ to a horizontal line.

compute a trajectoid $\mathfrak{C}$ that minimizes the function

$$\Psi(\mathfrak{T}) = \int_{t_0}^{t_\tau} \Xi(\mathfrak{T}, t)\, \mathrm{d}t.$$

So, at any time t, all entities lie in a disk of radius $\Xi(\mathfrak{C}, t)$ centered at $\mathfrak{C}(t)$.

Results and organization. We first study the situation where entities move in $\mathbb{R}^1$. In Section 7.1 we show that the worst-case complexity of a central trajectory in $\mathbb{R}^1$ is $\Theta(\tau n^2)$, and that we can compute one in $O(\tau n^2 \log n)$ time. We then extend our approach to entities moving in $\mathbb{R}^d$, for any constant d, in Section 7.2. For this case, we prove that the maximum complexity of a central trajectory $\mathfrak{C}$ is $O(\tau n^{5/2})$. Computing $\mathfrak{C}$ takes $O(\tau n^3)$ time and requires $O(\tau n^2 \log n)$ working space. We briefly discuss various extensions, such as using different centrality measures, in Section 7.3. Some of these complexities may seem fairly high, note however that in all cases the input size is $\Theta(\tau n)$. So, if τ and n are of the same order of magnitude all our results are at most quadratic in the input size.

7.1 Entities moving in $\mathbb{R}^1$

Let $\mathcal{E}$ be the set of entities moving in $\mathbb{R}^1$. The trajectories of these entities can be seen as polylines in $\mathbb{R}^2$: we associate time with the horizontal axis, and $\mathbb{R}^1$ with the vertical axis (see Figure 7.2). We observe that the distance between two points p and q in $\mathbb{R}^1$ is simply their absolute difference, that is, $\|pq\| = |p - q|$.

Let $\mathcal{I}$ be the *ideal* trajectory, that is, the trajectory that minimizes Ψ but is not restricted to lie on the input trajectories. It follows that at any time t, $\mathcal{I}(t)$ is simply the average of the highest entity $\mathcal{U}(t)$ and the lowest entity $\mathcal{L}(t)$. We further subdivide each time interval $J_i = [t_i, t_{i+1}]$ into *elementary intervals*, such that $\mathcal{I}$ is a single line segment.

▶ **Lemma 7.1.** *The total number of elementary intervals is* $\tau(n+2)$.

Proof. The ideal trajectory $\mathcal{I}$ changes direction when $\mathcal{U}(t)$ or $\mathcal{L}(t)$ changes. During a single interval $[t_i, t_{i+1}]$ all entities move along lines, so $\mathcal{U}$ and $\mathcal{L}$ are the upper and lower envelope of a set of n lines. So by standard point-line duality, $\mathcal{U}$ and $\mathcal{L}$ correspond to the upper and lower hull of n points. The summed complexity of the upper and lower hull is at most $n+2$. ☐

We transform the trajectories such that within each elementary interval $\mathcal{I}$ coincides with the x-axis. To simplify the description of the proofs and algorithms, we also assume that the entities never move parallel to the ideal trajectory, that is, there are no horizontal edges.

▶ **Lemma 7.2.** $\mathfrak{C}$ *is a central trajectory in* $\mathbb{R}^1$ *if and only if it minimizes the function*

$$\Psi'(\mathfrak{T}) = \int_{t_0}^{t_\tau} |\mathfrak{T}(t)| \, \mathrm{d}t.$$

Proof. A central trajectory $\mathfrak{C}$ is a trajectoid that minimizes the function

$$\begin{aligned}\Psi(\mathfrak{T}) &= \int_{t_0}^{t_\tau} \Xi(\mathfrak{T}, t) \, \mathrm{d}t = \int_{t_0}^{t_\tau} \max_{b \in \mathscr{E}} \|\mathfrak{T}(t) b(t)\| \, \mathrm{d}t = \int_{t_0}^{t_\tau} \max_{b \in \mathscr{E}} |\mathfrak{T}(t) - b(t)| \, \mathrm{d}t \\ &= \int_{t_0}^{t_\tau} \max\{|\mathfrak{T}(t) - \mathcal{U}(t)|, |\mathfrak{T}(t) - \mathcal{L}(t)|\} \, \mathrm{d}t.\end{aligned}$$

Since $(\mathcal{U}(t) + \mathcal{L}(t))/2 = 0$, we have that $|\mathfrak{T}(t) - \mathcal{U}(t)| > |\mathfrak{T}(t) - \mathcal{L}(t)|$ if and only if $\mathfrak{T}(t) < 0$. So, we split the integral, depending on $\mathfrak{T}(t)$, giving us

$$\begin{aligned}\Psi(\mathfrak{T}) &= \int_{t_0 \leq t \leq t_\tau \wedge \mathfrak{T}(t) \geq 0} \mathfrak{T}(t) - \mathcal{L}(t) \, \mathrm{d}t + \int_{t_0 \leq t \leq t_\tau \wedge \mathfrak{T}(t) < 0} \mathcal{U}(t) - \mathfrak{T}(t) \, \mathrm{d}t \\ &= \int_{t_0 \leq t \leq t_\tau \wedge \mathfrak{T}(t) \geq 0} \mathfrak{T}(t) \, \mathrm{d}t - \int_{t_0 \leq t \leq t_\tau \wedge \mathfrak{T}(t) \geq 0} \mathcal{L}(t) \, \mathrm{d}t + \\ &\quad \int_{t_0 \leq t \leq t_\tau \wedge \mathfrak{T}(t) < 0} \mathcal{U}(t) \, \mathrm{d}t - \int_{t_0 \leq t \leq t_\tau \wedge \mathfrak{T}(t) < 0} \mathfrak{T}(t) \, \mathrm{d}t.\end{aligned}$$

We now use that $-\int_{\mathfrak{T}(t)<0} \mathfrak{T}(t) = \int_{\mathfrak{T}(t)<0} |\mathfrak{T}(t)|$, and that $-\int \mathcal{L}(t) = \int \mathcal{U}(t)$ (since $(\mathcal{U}(t) + \mathcal{L}(t))/2 = 0$). After rearranging the terms we then obtain

$$\begin{aligned}\Psi(\mathfrak{T}) = &\int_{t_0 \leq t \leq t_\tau \wedge \mathfrak{T}(t) \geq 0} \mathfrak{T}(t)\,\mathrm{d}t + \int_{t_0 \leq t \leq t_\tau \wedge \mathfrak{T}(t) < 0} |\mathfrak{T}(t)|\,\mathrm{d}t + \\ &\int_{t_0 \leq t \leq t_\tau \wedge \mathfrak{T}(t) \geq 0} \mathcal{U}(t)\,\mathrm{d}t + \int_{t_0 \leq t \leq t_\tau \wedge \mathfrak{T}(t) < 0} \mathcal{U}(t)\,\mathrm{d}t \\ = &\int_{t_0 \leq t \leq t_\tau} |\mathfrak{T}(t)|\,\mathrm{d}t + \int_{t_0 \leq t \leq t_\tau} \mathcal{U}(t)\,\mathrm{d}t.\end{aligned}$$

The last term is independent of $\mathfrak{T}$, so we have $\Psi(\mathfrak{T}) = \Psi'(\mathfrak{T}) + c$, for some $c \in \mathbb{R}$. The lemma follows. ☐

By Lemma 7.2 a central trajectory $\mathfrak{C}$ is a trajectoid $\mathfrak{T}$ that minimizes the area $\Psi'(\mathfrak{T})$ between $\mathfrak{T}$ and the ideal trajectory $\mathcal{I}$. Hence, we can focus on finding a trajectoid that minimizes Ψ'.

7.1.1 Complexity

▶ **Lemma 7.3.** *A central trajectory $\mathfrak{C}$ for a set of n trajectories in $\mathbb{R}^1$, each with vertices at times $t_0, .., t_\tau$, may have worst-case complexity $\Omega(\tau n^2)$.*

Proof. We describe a construction for the entities that shows that within a single time interval $J = [t_i, t_{i+1}]$ the complexity of $\mathfrak{C}$ may be $\Omega(n^2)$. Repeating this construction $\Omega(\tau)$ times gives us $\Omega(\tau n^2)$ as desired.

Within J the entities move linearly. So we construct an arrangement $\mathcal{A}$ of lines that describes the motion of all entities. We place $m = n/3$ lines such that the upper envelope of $\mathcal{A}$ has linear complexity. We do the same for the lower envelope. We position these lines such that the ideal trajectory $\mathcal{I}$—which is the average of the upper and lower envelope— makes a vertical "zigzagging" pattern (see Figure 7.3). The remaining set H of m lines are horizontal. Two consecutive lines are placed at (vertical) distance at most ε. We place all lines such that they all intersect $\mathcal{I}$. It follows that $\mathfrak{C}$ jumps $\Omega(n^2)$ times between the lines in H (as is shown in Figure 7.3). The lemma follows. ☐

Two entities a and b are *ε-connected* at time t if there is a sequence $a = a_0, .., a_k = b$ of entities such that for all i, a_i and a_{i+1} are within distance ε of each other at time t. A subset $\mathcal{E}' \subseteq \mathcal{E}$ of entities is ε-connected at time t if all entities in $\mathcal{E}'$ are pairwise ε-connected at time t. The set $\mathcal{E}'$ is ε-connected during an interval I, if they are ε-connected at any time $t \in I$. We now observe:

▶ **Observation 7.4.** *$\mathfrak{C}$ can jump from entity a to b at time t if and only if a and b are ε-connected at time t.*

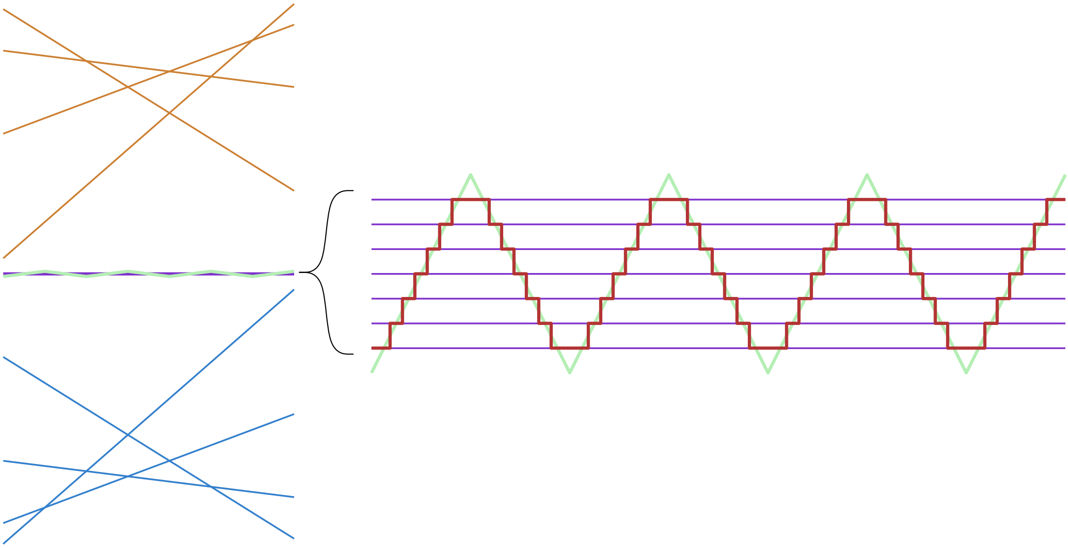


Figure 7.3: Lower bound construction that shows that $\mathfrak{C}$ (red) may have quadratic complexity. The ideal trajectory $\mathcal{I}$ is shown in green.

At any time t, we can partition $\mathcal{E}$ into maximal sets of ε-connected entities. The central trajectory $\mathfrak{C}$ must be in one of such maximal sets $\mathcal{E}'$: it uses the trajectory of an entity $a \in \mathcal{E}'$ (at time t), if and only if a is the entity from $\mathcal{E}'$ closest to $\mathcal{I}$. More formally, let $f_a(t) = |a(t)|$, and let $\mathcal{L}(\mathcal{F}) = \min_{f \in \mathcal{F}} f$ denote the lower envelope of a set of functions $\mathcal{F}$.

▶ **Observation 7.5.** *Let $\mathcal{E}'$ be a maximal set of entities that is ε-connected during interval J, let $a \in \mathcal{E}'$, and assume that $\mathfrak{C} \in \mathcal{E}'$ during J. For any time $t \in J$, we have that $\mathfrak{C}(t) = a(t)$ if and only if f_a is on the lower envelope of the set $\mathcal{F}' = \{f_b \mid b \in \mathcal{E}'\}$ at time t, that is, $f_a(t) = \mathcal{L}(\mathcal{F})(t)$.*

Let $\mathcal{E}_1, .., \mathcal{E}_m$ denote a collection of maximal sets of entities, such that $\mathcal{E}_i$ is ε-connected during time interval J_i. Let $\mathcal{F}_i = \{f_a \mid a \in \mathcal{E}_i\}$, and let $\mathcal{L}_i$ be the lower envelope $\mathcal{L}(\mathcal{F}_i)$ of $\mathcal{F}_i$ restricted to interval J_i. A lower envelope $\mathcal{L}_i$ has a break point at time t if $f_a(t) = f_b(t)$, for $a, b \in \mathcal{E}_i$. There are two types of break points: (i) $a(t) = b(t)$, or (ii) $a(t) = -b(t)$. At events of type (i) the modified trajectories of a and b intersect. At events of the type (ii), a and b are equally far from $\mathcal{I}$, but on different sides of $\mathcal{I}$. Let $B = \{(t, a, b) \mid \mathcal{L}_i(t) = f_a(t) = f_b(t) \wedge i \in \{1, .., m\}\}$ denote the collection of break points from all lower envelopes $\mathcal{L}_1, .., \mathcal{L}_m$.

▶ **Lemma 7.6.** *Consider a triplet* $(t, a, b) \in B$*. There is at most one lower envelope* $\mathcal{L}_i$ *such that* t *is a break point in* $\mathcal{L}_i$*.*

Proof. Assume by contradiction that t is a break point in both $\mathcal{L}_i$ and $\mathcal{L}_j$. At any time t, an entity can be in at most one maximal set $\mathcal{E}_\ell$. So if $\mathcal{E}_i$ and $\mathcal{E}_j$ share either entity a or b, then the intervals J_i and J_j are disjoint. It follows t cannot lie in both intervals, and thus cannot be a break point in both $\mathcal{L}_i$ and $\mathcal{L}_j$. Contradiction. ☐

▶ **Lemma 7.7.** *Let* $\mathcal{A}$ *be an arrangement of* n *lines, describing the movement of* n *entities during an elementary interval* J*. If there is a break point* $(t, a, b) \in B$*, with* $t \in J$*, of type (ii), then* $a(t)$ *and* $b(t)$ *lie on the boundary* $\partial\mathcal{Z}$ *of the zone* $\mathcal{Z}$ *of* $\mathcal{I}$ *in* $\mathcal{A}$*.*

Proof. Let $\mathcal{E}_j$ be the maximal ε-connected set containing a and b, and assume without loss of generality that $f_a(t) = a(t) = -b(t) = f_b(t)$. Now, assume by contradiction that a is not on $\partial\mathcal{Z}$ at time t (the case that $b(t)$ is not on $\partial\mathcal{Z}$ is symmetric). This means that there is an entity c with $0 \leq c(t) < a(t)$. If $c \in \mathcal{E}_j$, this contradicts that $f_a(t)$ was on the lower envelope of $\mathcal{E}_j$ at time t. So c is not ε-connected to a at time t. Hence, their distance is at least ε. We then have $a(t) > c(t) + \varepsilon > \varepsilon$. It now follows that a and b cannot be ε-connected at time t: the distance between a and b is bigger than ε so they are not directly connected, and f_a and f_b are on $\mathcal{L}_j$, so there are also no other entities in $\mathcal{E}_j$ through which they can be ε-connected. If a and b are not both in $\mathcal{E}_j$, they cannot contribute to a break point of $\mathcal{L}_j$. Contradiction. ☐

▶ **Lemma 7.8.** *Let* $\mathcal{A}$ *be an arrangement of* n *lines, describing the movement of* n *entities during an elementary interval* J*. The total number of break points* $(t, a, b) \in B$*, with* $t \in J$*, of type (ii) is at most* $6.5n$*.*

Proof. By Lemma 7.6 all break points can be charged to exactly one set $\mathcal{E}_j$. From Lemma 7.7 it follows that break points of type (ii) involve only entities whose lines in $\mathcal{A}$ participate in the zone of $\mathcal{I}$.

Let E be the set of edges of $\partial\mathcal{Z}$. We have that $|E| \leq 5.5n$ [18, 101]. We now split every edge that intersects $\mathcal{I}$, at the intersection point. Since every line intersects $\mathcal{I}$ at most once, this means the number of edges in E increases to $6.5n$. For every pair of edges (e, g), that lie on opposite sides of $\mathcal{I}$, there is at most one time t where a lower envelope $\mathcal{L}_j$, for some j, has a break point of type (ii).

Consider a break point of type (ii), that is, a time t such that $\mathcal{L} = \mathcal{L}_j$ switches (jumps) from an entity a to an entity b, with a and b on opposite sides of $\mathcal{I}$. Let $e \in E$ and $g \in E$ be the edges containing $a(t)$ and $b(t)$, respectively. If the arriving edge g has not been charged before, we charge the jump to g. Otherwise, we charge it to e. We continue to show that every edge in E is

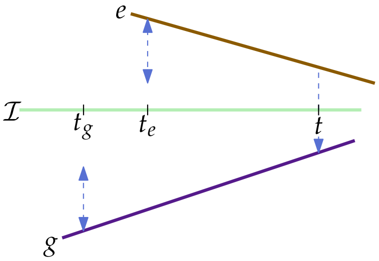


Figure 7.4: The jumps of $\mathcal{L}$ (dashed arrows) involving edges e and g. If g was charged by a jump at time $t_g < t_e < t$, then g was opposite to e at time t_e. Since $\mathcal{L}$ can jump at most once between e and g this yields a contradiction.

charged at most once. Since E has at most $6.5n$ edges, the number of break points of type (ii) is also at most $6.5n$.

We now show that either e or g has not been charged before. Assume, by contradiction, that both e and g have been charged before time t, at times t_e and t_g, respectively. Consider the case that $t_g < t_e$ (see Figure 7.4). At time t_e, the lower envelope $\mathcal{L}$ jumps from an edge h onto e or vice versa. Since there is a jump involving edge g at time t_g and one at time t it follows that at time t_e, g is the closest edge in E opposite to e. Hence, $h = g$. This means we jump twice between e and g. Contradiction. The case $t_e < t_g$ is symmetrical and the case $t_e = t_g$ cannot occur. It follows that e or g was not charged before time t, and thus all edges in E are charged at most once. ☐

▶ **Lemma 7.9.** *The total complexity of all lower envelopes* $\mathcal{L}_1, .., \mathcal{L}_m$ *on* $[t_i, t_{i+1}]$ *is* $O(n^2)$.

Proof. The break points in the lower envelopes are either of type (i) or of type (ii). We now show that there are at most $O(n^2)$ break points of either type.

The break points of type (i) correspond to intersections between the trajectories of two entities. Within interval $[t_i, t_{i+1}]$ the entities move along lines, hence there are at most $O(n^2)$ such intersections. By Lemma 7.6 all break points can be charged to exactly one set $\mathscr{E}_i$. It follows that the total number of break points of type (i) is $O(n^2)$.

To show that the number of events of the second type is at most $O(n^2)$ as well we divide $[t_i, t_{i+1}]$ in $O(n)$ elementary intervals such that $\mathcal{I}$ coincides with the x-axis. By Lemma 7.8 each such elementary interval contains at most $O(n)$ break points of type (ii). ☐

▶ **Theorem 7.10.** *Given a set of n trajectories in* $\mathbb{R}^1$*, each with vertices at times* $t_0, .., t_\tau$*, a central trajectory* $\mathfrak{C}$ *has worst case complexity* $O(\tau n^2)$.

Proof. A central trajectory $\mathfrak{C}$ is a piecewise function. From Observations 7.4 and 7.5 it now follows that $\mathfrak{C}$ has a break point at time t only if (a) two subsets of entities become ε-connected or ε-disconnected, or (b) the lower envelope of

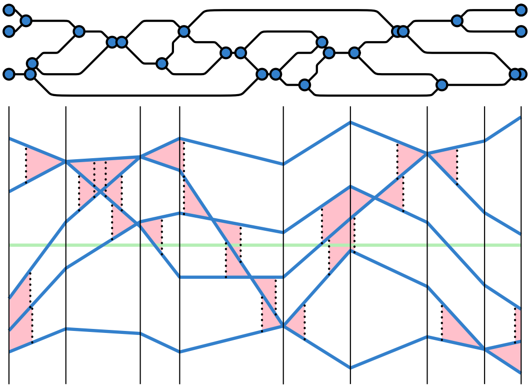


Figure 7.5: The Reeb graph for the moving entities from Figure 7.2. The dashed lines indicate that two entities are at distance ε.

a set of ε-connected entities has a break point at time t. Within a single time interval $J_i = [t_i, t_{i+1}]$ there are at most $O(n^2)$ times when two entities are at distance exactly ε. Hence, the number of events of type (a) during interval J_i is also $O(n^2)$. By Lemma 7.9 the total complexity of all lower envelopes of ε-connected sets during J_i is $O(n^2)$. Hence, the number of break points of type (b) within interval J_i is also $O(n^2)$. The theorem follows. ☐

7.1.2 Algorithm

We now present an algorithm to compute a trajectoid $\mathfrak{C}$ minimizing Ψ'. By Lemma 7.2 such a trajectoid is a central trajectory. The basic idea is to construct a weighted (directed acyclic) graph that represents a set of trajectoids containing $\mathfrak{C}$. We can then find $\mathfrak{C}$ by computing a minimum weight path in this graph.

The graph that we use is a weighted version of the Reeb graph presented in Section 5.1 of Chapter 5. Each edge $e = (u, v)$ of the Reeb graph $\mathcal{R}$ corresponds to a maximal subset of entities $C_e \subseteq \mathcal{E}$ that is ε-connected during the time interval $[t_u, t_v]$. The vertices represent times at which the sets of ε-connected entities change, that is, the times at which two entities a and b are at distance ε from each other and the set containing a merges with or splits from the set containing b. See Figure 7.5 for an illustration.

By Observation 7.4 a central trajectory $\mathfrak{C}$ can jump from a to b if and only if a and b are ε-connected, that is, if a and b are in the same component C_e of edge e. From Observation 7.5 it follows that on each edge e, $\mathfrak{C}$ uses only the trajectories of entities a for which f_a occurs on the lower envelope of the functions $\mathcal{F}_e = \{f_a \mid a \in C_e\}$. Hence, the cost of using edge e is

$$\omega_e = \int_{t_u}^{t_v} \mathcal{L}(\mathcal{F}_e)(t)\,\mathrm{d}t.$$

Thus, $\mathfrak{C}$ follows a path in the Reeb graph $\mathcal{R}$. In other words, the set of trajectoids represented by $\mathcal{R}$ contains a trajectoid minimizing Ψ'. We can compute a central trajectory by finding a minimum weight path in $\mathcal{R}$ from a source to a sink.

Analysis. First we compute the Reeb graph using the algorithm from Section 5.2.1. This takes $O(\tau n^2 \log n)$ time. Second we compute the weight ω_e for each edge e. The Reeb graph $\mathcal{R}$ is a DAG, so once we have the edge weights, we can use dynamic programming to compute a minimum weight path in $O(|\mathcal{R}|) = O(\tau n^2)$ time. So all that remains is to compute the edge weights ω_e. For this, we need the lower envelope $\mathcal{L}_e$ of each set $\mathcal{F}_e$ on the interval J_e. To compute the lower envelopes, we need the ideal trajectory $\mathcal{I}$. We can compute $\mathcal{I}$ in $O(\tau n \log n)$ time by computing the lower and upper envelope of the trajectories in each time interval $[t_i, t_{i+1}]$.

Lemma 7.9 implies that the total complexity of all lower envelopes is $O(\tau n^2)$. To compute them we have two options. We can simply compute the lower envelope from scratch for every edge of $\mathcal{R}$. This takes $O(\tau n^2 \cdot n \log n) = O(\tau n^3 \log n)$ time. Alternatively, for each time interval $J_i = [t_i, t_{i+1}]$, we compute the arrangement $\mathcal{A}$ representing the modified trajectories on the interval J_i, and use it to trace $\mathcal{L}_e$ in $\mathcal{A}$ for every edge e of $\mathcal{R}$.

Using a standard sweep line algorithm, an arrangement of m line segments can be built in $O((m + A) \log m)$ time, where A is the output complexity. We have $O(n^2)$ line segments: $n + 2$ per entity. Since each pair of trajectories intersects at most once during J_i, we have that $A = O(n^2)$. Thus, we can build $\mathcal{A}$ in $O(n^2 \log n)$ time. The arrangement $\mathcal{A}$ represents all break points of type (i), of all functions f_a. We now compute all pairs of points in $\mathcal{A}$ corresponding to break points of type (ii). We do this in $O(n^2)$ time by traversing the zone of $\mathcal{I}$ in $\mathcal{A}$.

We can then trace the lower envelopes through $\mathcal{A}$: for each edge $e = (u, v)$ in the Reeb graph with $J_e \subseteq J_i$, we start at the point $a(t_u)$, $a \in C_e$, that is closest to $\mathcal{I}$, and then follow the edges in $\mathcal{A}$ corresponding to $\mathcal{L}_e$, taking care to jump when we encounter break points of type (ii). Our lower envelopes are all disjoint (except at endpoints), so we traverse each edge in $\mathcal{A}$ at most once. The same holds for the jumps. We can avoid costs for searching for the starting point of each lower envelope by tracing the lower envelopes in the right order: when we are done tracing $\mathcal{L}_e$, with $e = (u, v)$, we continue with the lower envelope of an outgoing edge of vertex v. If v is a split vertex where a and b are at distance ε, then the starting point of the lower envelope of the other edge is either $a(t_v)$ or $b(t_v)$, depending on which of the two is farthest from $\mathcal{I}$. It follows that when we have $\mathcal{A}$ and the list of break points of type (ii), we can compute all lower envelopes in $O(n^2)$ time. We conclude:

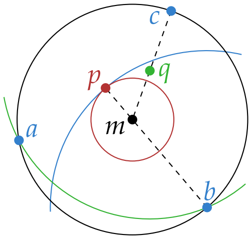


Figure 7.6: Point p is closest to the ideal point m, however the smallest enclosing disk centered at q is smaller than that of p.

▶ **Theorem 7.11.** *Given a set of n trajectories in $\mathbb{R}^1$, each with vertices at times $t_0, .., t_\tau$, we can compute a central trajectory $\mathfrak{C}$ in $O(\tau n^2 \log n)$ time using $O(\tau n^2)$ space.*

7.2 Entities moving in $\mathbb{R}^d$

In the previous section, we used the ideal trajectory $\mathcal{I}$, which minimizes the distance to the farthest entity, ignoring the requirement to stay on an input trajectory. The problem was then equivalent to finding a trajectoid that minimizes the distance to the ideal trajectory. In $\mathbb{R}^d$, with $d > 1$, however, this approach fails, as the following example shows.

▶ **Observation 7.12.** *Let P be a set of points in $\mathbb{R}^2$. The point in P that minimizes the distance to the ideal point (i.e., the center of the smallest enclosing disk of P) is not necessarily the same as the point in P that minimizes the distance to the farthest point in P.*

Proof. See Figure 7.6. Consider three points a, b and c at the corners of an equilateral triangle, and two points p and q close to the center m of the circle through a, b and c. Now p is closer to m than q, yet q is closer to b than p, and q is as far from a as from b. ☐

7.2.1 Complexity

It follows from Lemma 7.3 that the complexity of a central trajectory for entities moving in $\mathbb{R}^d$ is at least $\Omega(\tau n^2)$. In this section, we prove that the complexity of $\mathfrak{C}$ within a single time interval $[t_i, t_{i+1}]$ is at most $O(n^{5/2})$. Thus, the complexity over all τ time intervals is $O(\tau n^{5/2})$.

Recall that Ξ_a is the function expressing the distance from a to the entity farthest away from a. Let $\mathcal{F}$ denote the collection of functions Ξ_a, for $a \in \mathscr{E}$. We partition time into intervals $J'_1, .., J'_{k'}$ such that in each interval J'_i all functions Ξ_a restricted to J'_i are *simple*, that is, they consist of just one piece. We now show that each function Ξ_a consists of at most $\tau(2n-1)$ pieces, and thus the total number of intervals is at most $O(\tau n^2)$. See Figure 7.1(c) for an illustration.

▶ **Lemma 7.13.** *Each function Ξ_a is piecewise hyperbolic and consists of at most $\tau(2n-1)$ pieces.*

Proof. Consider a time interval $J_i = [t_i, t_{i+1}]$. In interval J_i, the function ξ_{ab}, with $b \in \mathscr{E}$, is a convex hyperbolic function (see the discussion in Section 6.1). Each pair of such functions can intersect at most twice. During J_i, Ξ_a is the upper envelope of these functions ξ_{ab}, so it consists of $\lambda_2(n)$ pieces, where λ_s denotes the maximum complexity of a Davenport-Schinzel sequence of order s [2]. We have $\lambda_2(n) = 2n-1$, so the lemma follows. ☐

▶ **Lemma 7.14.** *The total number of intersections of all functions in $\mathcal{F}$ is at most $O(\tau n^3)$.*

Proof. Fix a pair of entities a, b. By Lemma 7.13 there are at most $\tau(2n-1)$ time intervals J, such that Ξ_a restricted to J is simple. The same holds for Ξ_b. So, there are at most $\tau(4n-2)$ intervals in which both Ξ_a and Ξ_b are simple (and hyperbolic). In each interval Ξ_a and Ξ_b intersect at most twice. ☐

We again observe that $\mathfrak{C}$ can jump from one entity to another only if they are ε-connected. Hence, Observation 7.4 holds for entities moving in $\mathbb{R}^d$ as well. As before, this means that at any time t, we can partition $\mathscr{E}$ into maximal sets of ε-connected entities. Let $\mathscr{E}' \ni a$ be a maximal subset of ε-connected entities at time t. This time, a central trajectory $\mathfrak{C}$ uses the trajectory of entity a at time t, if and only if a is the entity from $\mathscr{E}'$ whose function Ξ_a is minimal. Hence, if we define $f_a = \Xi_a$, then Observation 7.5 holds again as well.

Consider all $m' = O(n^2)$ intervals $J'_1, .., J'_{m'}$ that together form $[t_j, t_{j+1}]$. We subdivide these intervals at points where the distance between two entities is exactly ε. Let $J_1, .., J_m$ denote the set of resulting intervals. Since there are $O(n^2)$ times at which two entities are at distance exactly ε, we still have $O(n^2)$ intervals. Note that for all intervals J_i and all entities a, f_a is simple and totally defined on J_i.

In each interval J_i, a central trajectory $\mathfrak{C}$ uses the trajectories of only one maximal set of ε-connected entities. Let $\mathscr{E}'_i$ be this set, let $\mathcal{F}'_i = \{f_a \mid a \in \mathscr{E}'_i\}$ be the set of corresponding functions, and let $\mathcal{L}_i$ be the lower envelope of $\mathcal{F}'_i$, restricted to interval J_i. We now show that the total complexity of all these lower envelopes is $O(n^{5/2})$. It follows that the maximal complexity of $\mathfrak{C}$ in J_i is at most $O(n^{5/2})$ as well.

▶ **Lemma 7.15.** *Let J be an interval, let $\mathcal{F}$ be a set of hyperbolic functions that are simple and total on J, and let k denote the complexity of the lower envelope $\mathcal{L}$ of $\mathcal{F}$ restricted to J. There are $\Omega(k^2)$ intersections of functions in $\mathcal{F}$ that do not lie on $\mathcal{L}$.*

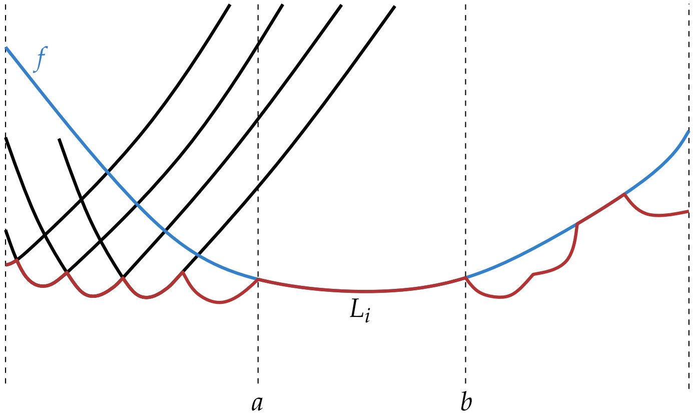


Figure 7.7: The function f (blue) has at least $\ell_{i-2} = \lfloor(i-1)/2\rfloor$ intersections with functions from $\mathcal{F}_i$ (black).

Proof. Let $\mathcal{L} = L_1, .., L_k$ denote the pieces of the lower envelope, ordered from left to right. Consider any prefix $\mathcal{L}' = L_1, ..., L_i$ of the pieces. The functions in $\mathcal{F}$ are all hyperbolic, so every pair of functions intersects at most twice. Therefore any $\mathcal{L}'$ consists of at most $\lambda_2(|\mathcal{F}|) = 2|\mathcal{F}| - 1$ pieces. Hence, $i \leq 2|\mathcal{F}| - 1$. The same argument gives us that there must be at least $\ell_i = \lfloor(i+1)/2\rfloor$ distinct functions of $\mathcal{F}$ contributing to $\mathcal{L}'$.

Consider a piece $L_i = [a, b]$ such that a is the first time that a function f contributes to the lower envelope. That is, a is the first time such that $f(t) = \mathcal{L}(t)$. Clearly, there are at least ℓ_k such pieces. Furthermore, there are at least ℓ_{i-2} distinct functions corresponding to the pieces $L_1, .., L_{i-2}$. Let $\mathcal{F}_i$ denote the set of those functions.

All functions in $\mathcal{F}$ are continuous and total, so they span time interval J. It follows that all functions in $\mathcal{F}_i$ must intersect f at some time after the start of interval J, and before time a. Since a was the first time that f lies on $\mathcal{L}$, all these intersection points do not lie on $\mathcal{L}$. See Figure 7.7. In total we have at least

$$\sum_{i=2}^{\ell_k} \ell_{i-2} = \sum_{i=2}^{\lfloor(k+1)/2\rfloor} \lfloor(i-1)/2\rfloor = \sum_{i=1}^{\lfloor(k+1)/2\rfloor-1} \lfloor i/2\rfloor = \Omega(k^2)$$

such intersections. □

▶ **Lemma 7.16.** *Let $\mathcal{F}_1, .., \mathcal{F}_m$ be a collection of m sets of partial functions, let $J_1, .., J_m$ be a collection of intervals such that:*

- *the total number of intersections between functions in $\mathcal{F}_1, .., \mathcal{F}_m$ is at most $O(n^3)$,*
- *for any two intersecting intervals J_i and J_j, $\mathcal{F}_i$ and $\mathcal{F}_j$ are disjoint, and*
- *for every set $\mathcal{F}_i$, all functions in $\mathcal{F}_i$ are simple, hyperbolic, and totally defined on J_i.*

Let $\mathcal{L}_i$ denote the lower envelope of $\mathcal{F}_i$ restricted to J_i. The total complexity of the lower envelopes $\mathcal{L}_1, .., \mathcal{L}_m$ is $O((m + n^2)\sqrt{n})$.

Proof. Let k_i denote the complexity of the lower envelope $\mathcal{L}_i$. An interval J_i is *heavy* if $k_i > \sqrt{n}$ and *light* otherwise. Clearly, the total complexity of all light intervals is at most $O(m\sqrt{n})$. What remains is to bound the complexity of all heavy intervals.

Relabel the intervals such that $J_1, .., J_h$ are the heavy intervals. By Lemma 7.15 we have that in each interval J_i, there are at least ck_i^2 intersections involving the functions $\mathcal{F}_i$, for some $c \in \mathbb{R}$.

Since for every pair of intervals J_i and J_j that overlap the sets $\mathcal{F}_i$ and $\mathcal{F}_j$ are disjoint, we can associate each intersection with at most one interval. There are at most $O(n^3)$ intersections in total, thus we have $c'n^3 \geq \sum_{i=1}^{m} ck_i^2 \geq \sum_{i=1}^{h} ck_i^2$, for some $c' \in \mathbb{R}$. Using that for all heavy intervals $k_i > \sqrt{n}$ we obtain

$$c'n^3 \geq \sum_{i=1}^{h} ck_i^2 \geq \sum_{i=1}^{h} c\sqrt{n}k_i = c\sqrt{n}\sum_{i=1}^{h} k_i.$$

It follows that the total complexity of the heavy intervals is

$$\sum_{i=1}^{h} k_i \leq c'n^3/c\sqrt{n} = O(n^2\sqrt{n}).$$

□

By Lemma 7.14 we have that the number of intersections between functions in $\mathcal{F}$ in time interval $[t_j, t_{j+1}]$ is $O(n^3)$. Hence, the total number of intersections over all functions in all sets $\mathcal{F}'_i$ is also $O(n^3)$. All functions in each set $\mathcal{F}'_i$ are simple and totally defined on J_i, and all intervals $J_1, .., J_m$ are pairwise disjoint, so we can use Lemma 7.16. It follows that the total complexity of $\mathcal{L}'_1, .., \mathcal{L}'_m$ is at most $O(n^{5/2})$. Thus, in a single time interval the worst case complexity of $\mathfrak{C}$ is also at most $O(n^{5/2})$. We conclude:

▶ **Theorem 7.17.** *Given a set of n trajectories in $\mathbb{R}^d$, each with vertices at times $t_0, .., t_\tau$, a central trajectory $\mathfrak{C}$ has worst case complexity $O(\tau n^{5/2})$.*

7.2.2 Algorithm

We use the same global approach as before: we represent a set of trajectoids containing an optimal solution by a graph, and then compute a minimum weight path in this graph. The graph that we use, is a slightly modified Reeb graph. We split an edge e into two edges at time t if there is an entity $a \in C_e$

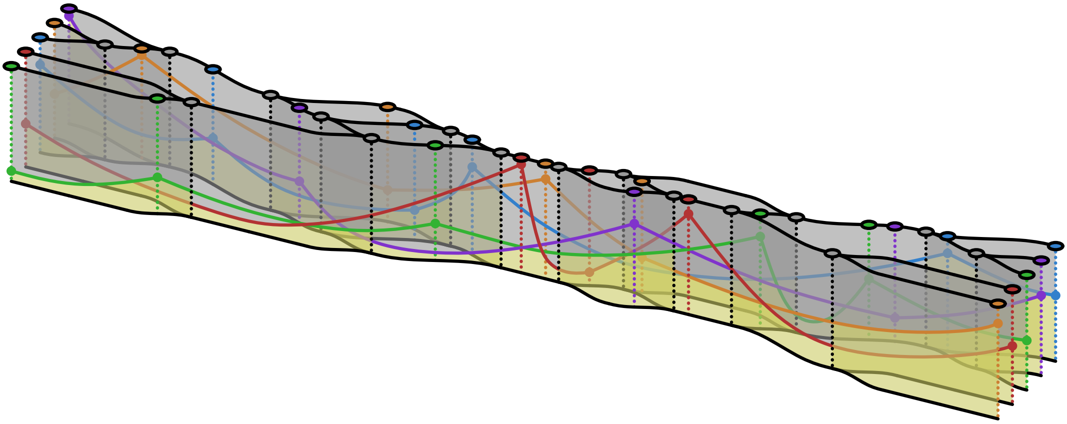


Figure 7.8: The modified Reeb graph $\mathcal{R}$ for five entities moving during a single interval, and the corresponding functions f_a for each entity a.

such that $\Xi_a = f_a$ has a break point at time t. All functions f_a, with $a \in C_e$, are now simple and totally defined on J_e. This process adds a total of $O(\tau n^2)$ degree-two vertices to the Reeb graph. Let $\mathcal{R}$ denote the resulting Reeb graph (see Figure 7.8).

To find all the times where we have to insert vertices, we explicitly compute the functions Ξ_a. This takes $O(\tau n \lambda_2(n) \log n) = O(\tau n^2 \log n)$ time, where λ_s denotes the maximum length of a Davenport-Schinzel sequence of order s [2], since within each time interval $[t_i, t_{i+1}]$ each Ξ_a is the upper envelope of a set of n functions that intersect each other at most twice. After we sort these break points in $O(\tau n^2 \log n)$ time, we can compute the modified Reeb graph $\mathcal{R}$ in $O(\tau n^2 \log n)$ time (Section 5.2.1).

Next, we compute all weights ω_e, for each edge e. This means we have to compute the lower envelope $\mathcal{L}_e$ of the functions $\mathcal{F}_e = \{f_a \mid a \in C_e\}$ on the interval J_e. All these lower envelopes have a total complexity of at most $O(\tau n^{5/2})$:

▶ **Lemma 7.18.** *The total complexity of the lower envelopes for all edges of the Reeb graph is $O(\tau n^{5/2})$.*

Proof. We consider each time interval $J_i = [t_i, t_{i+1}]$ separately. Let $\mathcal{R}_i$ denote the Reeb graph restricted to J_i. We now show that for each $\mathcal{R}_i$, the total complexity of all lower envelopes $\mathcal{L}_e$ of edges e in $\mathcal{R}_i$ is $O(n^2\sqrt{n})$. Since there are $O(\tau)$ time intervals J_i, the lemma then follows.

By Lemma 7.14, the total number of intersections of all functions $\mathcal{F}_e$, with e in $\mathcal{R}_i$, is $O(n^3)$. Each set $\mathcal{F}_e$ corresponds to an interval J_e, on which all functions in $\mathcal{F}_e$ are simple and totally defined. Furthermore, at any time, every entity is in at most one component C_e. So, if two intervals J_e and $J_{e'}$ overlap, the sets of entities C_e and $C_{e'}$, and thus also the sets of functions $\mathcal{F}_e$ and $\mathcal{F}_{e'}$ are disjoint. It follows that we can apply Lemma 7.16. Since $\mathcal{R}_i$ has $O(n^2)$ edges the total complexity of all lower envelopes is $O(n^2\sqrt{n})$. □

We again have two options to compute all lower envelopes: either we compute all of them from scratch in $O(\tau n^2 \cdot \lambda_2(n) \log n) = O(\tau n^3 \log n)$ time, or we use a similar approach as before. For each time interval, we compute the arrangement $\mathcal{A}$ of all functions $\mathcal{F}$, and then trace $\mathcal{L}_e$ in $\mathcal{A}$ for every edge e. For n^2 functions that pairwise intersect at most twice, the arrangement can be built in $O(n^2 \log n + A)$ time, where A is the output complexity [8]. The complexity of $\mathcal{A}$ is $O(n^3)$, so we can construct it in $O(n^3)$ time. As before, every edge is traversed at most once so tracing all lower envelopes $\mathcal{L}_e$ takes at most $O(n^3)$ time. It follows that we can compute all edge weights in $O(\tau n^3)$ time, using $O(n^3)$ working space.

Computing a minimum weight path takes $O(\tau n^2)$ time, and uses $O(\tau n^2)$ space as before. Thus, we can compute $\mathfrak{C}$ in $O(\tau n^3)$ time and $O(n^3 + \tau n^2)$ space.

Reducing the required working space. We can reduce the amount of working space required to $O(n^2 \log n + \tau n^2)$ as follows. Consider computing the edge weights in the time interval $J = [t_i, t_{i+1}]$. Interval J is subdivided into $O(n^2)$ smaller intervals $J_1, .., J_m$ as described in Section 7.2.1. We now consider groups of r consecutive intervals. Let J be the union of r consecutive intervals, we compute the arrangement $\mathcal{A}$ of the functions $\mathcal{F}$, restricted to time interval J. Since every interval J_i has at most $O(n^2)$ intersections, $\mathcal{A}$ has worst-case complexity $O(rn^2)$. Thus, at any time we need at most $O(rn^2)$ space to store the arrangement. In total this takes $O(\sum_{i=1}^{n^2/r}(n_i \log n_i + A_i))$ time, where n_i is the number of functions in the i^{th} group of intervals, and A_i is the complexity of the arrangement in group i. The total complexity of all arrangements is again $O(n^3)$. Since we cut each function Ξ_a into an additional $O(n^2/r)$ pieces, the total number of functions is $O(n^3/r + n^2)$. Hence, the total running time is $O((n^3/r)\log n + n^3)$. We now choose $r = \Theta(\log n)$ to compute all edge weights in $[t_i, t_{i+1}]$ in $O(n^3)$ time and $O(n^2 \log n)$ space. We conclude:

▶ **Theorem 7.19.** *Given a set of n trajectories in $\mathbb{R}^d$, each with vertices at times $t_0, .., t_\tau$, we can compute a central trajectory $\mathfrak{C}$ in $O(\tau n^3)$ time using $O(n^2 \log n + \tau n^2)$ space.*

7.3 Extensions

We now briefly discuss how our results can be extended in various directions.

Other measures of centrality. We based our central trajectory on the center of the smallest enclosing disk of a set of points. Instead, we could choose other static measures of centrality, such as the Fermat-Weber point, which minimizes the sum of distances to the other points, or the center of mass, which minimizes the sum of squared distances to the other points. In both cases we can use the same general approach as described in Section 7.2.

Let $\hat{\Xi}_a^2(t) = \sum_{b \in \mathscr{E}} \zeta_{ab}(t)^2$ denote the sum of the squared Euclidean distances from a to all other entities at time t. This function $\hat{\Xi}_a^2$ is piecewise quadratic in t, and consists of (only) $O(\tau)$ pieces. It follows that the total number of intersections between all functions $\hat{\Xi}_a^2$, $a \in \mathscr{E}$, is at most $O(\tau n^2)$. We again split the domain of these functions into elementary intervals. The Reeb graph $\mathcal{R}$ representing the ε-connectivity of the entities still has $O(\tau n^2)$ vertices and edges. Each vertex of $\mathfrak{C}$ corresponds either to an intersection between two functions $\hat{\Xi}_a^2$ and $\hat{\Xi}_b^2$, or to a jump, occurring at a vertex of $\mathcal{R}$. It now follows that $\mathfrak{C}$ has complexity $O(\tau n^2)$.

To compute a central trajectory, we compute a shortest path in the (weighted) Reeb graph $\mathcal{R}$. To compute the weights we again construct the arrangement of all curves $\hat{\Xi}_a^2$, and trace the lower envelope $\mathcal{L}_e$ of the curves associated to each edge $e \in \mathcal{R}$. This can be done in $O(\tau n^2 \log n)$ time in total.

The sum of Euclidean distances $\hat{\Xi}_a(t) = \sum_{b \in \mathscr{E}} \|a(t)b(t)\|$ is a sum of square roots, and cannot be represented analytically in an efficient manner.[1] Hence, we cannot efficiently compute a central trajectory for this measure.

Similarly, depending on the application, we may prefer a different way of integrating over time. Instead of the integral of Ξ, we may, for example, wish to minimize $\max_t \Xi(\cdot, t)$ or $\int \Xi^2(\cdot, t)\,\mathrm{d}t$. Again, the same general approach still works, but now, after constructing the Reeb graph, we compute the weights of each edge differently.

Minimizing the distance to the ideal trajectory $\mathcal{I}$. We saw that for entities moving in $\mathbb{R}^1$, minimizing the distance from $\mathfrak{C}$ to the farthest entity is identical to minimizing the distance from $\mathfrak{C}$ to the ideal trajectory $\mathcal{I}$ (which itself minimizes the distance to the farthest entity, but is not constrained to lie on an input trajectory). We also saw that for entities moving in $\mathbb{R}^d$, $d > 1$, these

---

[1] This is the problem we encountered in Chapter 4 when we wanted to find hotspots with curved boundaries.

two problems are not the same. So, a natural question is whether we can also minimize the distance to $\mathcal{I}$ in this case. It turns out that, at least for $\mathbb{R}^2$, we can again use our general approach, albeit with different complexities.

Demaine et al. [38] show that for entities moving along lines[2] in $\mathbb{R}^2$ the ideal trajectory $\mathcal{I}$ has complexity $O(n^{3+\delta})$ for any $\delta > 0$. It follows that the function $\breve{\Xi}_a(t) = \|\mathcal{I}(t)a(t)\|$ is a piecewise hyperbolic function with at most $O(\tau n^{3+\delta})$ pieces. The total number of intersections between all functions $\breve{\Xi}_a$, for $a \in \mathscr{E}$, is then $O(\tau n^{5+\delta})$. Similar to Lemma 7.16, we can then show that all lower envelopes in $\mathcal{R}$ together have complexity $O(\tau n^{4+\delta})$. We then also obtain an $O(\tau n^{4+\delta})$ bound on the complexity of a central trajectory $\mathfrak{C}$ minimizing the distance to $\mathcal{I}$.

To compute such a central trajectory $\mathfrak{C}$ we again construct $\mathcal{R}$. To compute the edge weights it is now more efficient to recompute the lower envelope $\mathcal{L}_e$ for each edge e from scratch. This takes $O(\tau n^3 \cdot n \log n) = O(\tau n^4 \log n)$ time, whereas constructing the entire arrangement may take up to $O(\tau n^{5+\delta})$ time.

We note that the $O(n^{3+\delta})$ bound on the complexity of $\mathcal{I}$ by Demaine et al. [38] is not known to be tight. The best known lower bound is only $\Omega(n^2)$. So, a better upper bound for this problem also gives a better bound on the complexity of $\mathfrak{C}$ and on the running time of our algorithm.

Relaxing the input pieces requirement. We require each piece of the central trajectory to be part of one of the input trajectories, and allow small jumps between the trajectories. This is necessary, because in general no two trajectories may intersect. Another interpretation of this dilemma is to relax the requirement that the output trajectory stays on an input trajectory at all times, and just require it to be *close* (within distance ε) to an input trajectory at all times. In this case, no discontinuities in the output trajectory are necessary.

We can model this by replacing each point entity by a disk of radius ε. The goal is then to compute a path that stays within the union of disks at all times, minimizes Ψ. We now observe that if at time t the ideal trajectory $\mathcal{I}$ is contained in the same component of ε-disks as $\mathfrak{C}$, the central trajectory will follow $\mathcal{I}$. If $\mathcal{I}$ lies outside of the component, $\mathfrak{C}$ travels on the border of the ε-disk (in the component containing $\mathfrak{C}$) minimizing $\Xi(\cdot, t)$. In terms of the distance functions, this behavior again corresponds to following the lower envelope of a set of functions. We can thus identify the following types of break points of $\mathfrak{C}$: (i) break points of $\mathcal{I}$, (ii) breakpoints in one of the lower envelopes $\mathcal{L}_1, .., \mathcal{L}_m$ corresponding to the distance functions of the entities in each component, and (iii) break points at which $\mathfrak{C}$ switches between following $\mathcal{I}$ and following a lower envelope $\mathcal{L}_j$. There are at most $O(\tau n^{3+\delta})$ break points of type (i) [38],

---

[2] Or, more generally, along a curve described by a low degree polynomial.

and at most $O(\tau n^2 \sqrt{n})$ of type (ii). The break points of type (iii) correspond to intersections between $\mathcal{I}$ and the manifold that we get by tracing the ε-disks over the trajectory. The number of such intersections is at most $O(\tau n^{4+\delta})$. Hence, in this case $\mathfrak{C}$ has complexity $O(\tau n^{4+\delta})$. We can thus get an $O(\tau n^{5+\delta} \log n)$ algorithm by computing the lower envelopes from scratch.

7.4 Concluding Remarks

We considered the problem of computing a time-dependent representative trajectory for a given set of trajectories. We introduced such a representative: a central trajectory, which consists of pieces of the input trajectories, jumps from one trajectory to another only if they are close together, and minimizes the size of the smallest enclosing disk over time. In the situation where the entities generating the trajectories move in $\mathbb{R}^1$, we showed that the worst case complexity of a central trajectory is $\Theta(\tau n^2)$, and that we can compute one in $O(\tau n^2 \log n)$ time. We then extended our approach to entities moving in $\mathbb{R}^d$, for any constant d. In this case we proved that the maximal complexity of a central trajectory is $O(\tau n^{5/2})$, and that it can be computed in $O(\tau n^3)$ time using $O(n^2 \log n + \tau n^2)$ working space.

Even though we do not expect this to happen in practice, the worst case complexity of our central trajectories can be higher than the input size. If this occurs, we can use traditional line simplification algorithms like Imai and Iri [73] to simplify the resulting central trajectory. This gives us a representative that still is always close —for instance within distance 2ε— to one of the input trajectories. Alternatively, we can use dynamic-programming combined with our methods to enforce the output trajectory to have at most k vertices, for any k, and always be on the input trajectories. Computing such a central trajectory is more expensive than our current algorithms, however. Furthermore, enforcing a low output complexity may not be necessary. For example, in applications like visualization, the number of trajectories shown often has a larger impact visual clutter than the length or complexity of the individual trajectories. It may be easier to follow a single trajectory that has many vertices than to follow many trajectories that have fewer vertices each.

There are various interesting open problems. First, there is still a gap between the lower and upper bound on the complexity of a central trajectory in $\mathbb{R}^d$, with $d \geq 2$. It would be interesting to close this gap. Second, we would like an output-sensitive algorithm to compute a central trajectory for entities moving in $\mathbb{R}^d$. In our current approach we need the lower envelope of a set of functions $\mathcal{F}_e$ for edge e of the Reeb graph. For two such edges e and f, the sets $\mathcal{F}_e$ and

$\mathcal{F}_f$ may have a large number of functions in common, however we currently do not have a good way to use this to our advantage. More concretely, we would like a data structure that can maintain the lower envelope of a set of functions $\mathcal{F}$, and allows us to efficiently merge $\mathcal{F}$ with another set $\mathcal{G}$, or split it into sets $\mathcal{F}_1$ and $\mathcal{F}_2$. Third, from a modeling perspective, the relaxed model in which the central trajectory stays within the union of "tubes" that we describe at the end of Section 7.3 seems most natural. However, our bound on the complexity of a central trajectory in that case is rather large. It would be very interesting to improve on this.

Part IV

Concluding Remarks

Chapter 8

Concluding Remarks

Technology such as the Global Positing System (GPS) has made tracking moving entities easy and cheap. As a result there is a large amount of trajectory data available, and an increasing demand for tools and techniques to analyze such data. We considered four analysis tasks for trajectory data, and developed efficient algorithms to perform them automatically. Two of these tasks involved a single trajectory, and two of which involved multiple trajectories. In each case, we used an approach typical in computational geometry: we formalized the problem at hand, and analyzed its geometric properties. We then used these properties to obtain efficient algorithms, often by using and developing interesting techniques along the way.

In Chapter 3 we studied the problem of finding a *segmentation* of a trajectory based on a non-monotone criterion. A segmentation is a partition of the trajectory into (maximal) segments (contiguous sub-trajectories) such that the given criterion holds on each segment. For a class of criteria, monotone criteria, it was known how to compute a segmentation efficiently. We analyzed how to handle non-monotone criteria, and presented a solution that can also handle non-monotone criteria, provided they satisfy certain properties.

In Chapter 4 we studied the problem of finding *hotspots*; regions in which the entity generating the trajectory spent a large amount of time. Several versions of the problem exist. Two examples are (i) given the desired size of the hotspot, find the location of a hotspot of that size that maximizes the time the entity spends inside it, and (ii) given an amount of time L, find a smallest hotspot in which the entity stays at least L time units. We provided efficient algorithms for six such versions, all of which easily extend to handling multiple trajectories.

In Chapters 5 and 6 we studied the problem of finding all (maximal) groups and the *trajectory grouping structure*. A group is a movement pattern in which sufficiently many entities move together during a sufficiently long time interval. In addition to the groups themselves we also found the relation between groups, e.g. a large group came into existence when two smaller groups merged. To this end, we used a topological structure called the *Reeb graph*. In Chapter 5

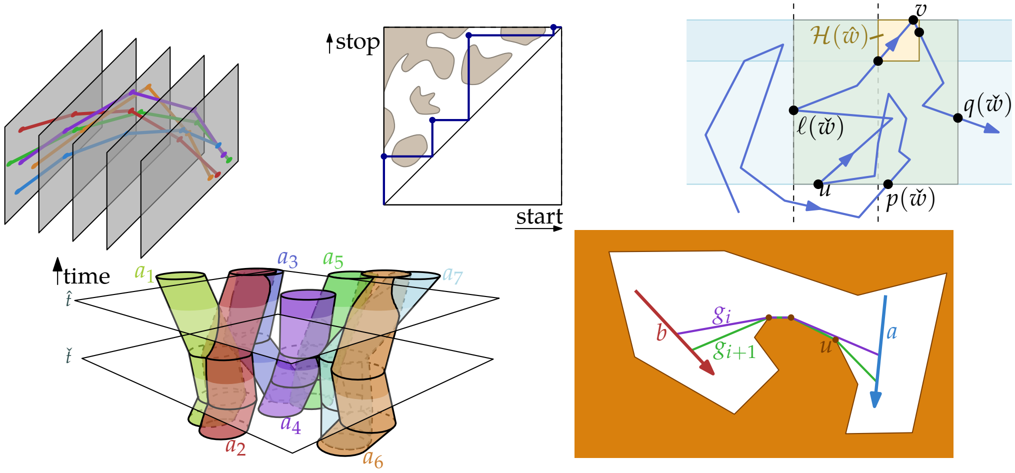


Figure 8.1: Some of the ingredients used in the work presented in this thesis.

we presented the algorithms to compute the Reeb graph and described how to compute the trajectory grouping structure, and all maximal groups using the Reeb graph. In Chapter 6 we significantly extended our results to the case where the entities generating the trajectories move in a space containing obstacles.

Finally, in Chapter 7 we studied the problem of finding a representative for a set of trajectories. The representative should capture the defining features of all trajectories in the input set, and incorporate both the spatial and the temporal component of the trajectories. We introduced such a representative, the *central trajectory*, and showed that it has these properties. Additionally, we provided an efficient algorithm to compute a central trajectory by using the Reeb graph from Chapter 5.

8.1 Outlook

The work in this thesis presents a significant step forward in the field of trajectory and movement analysis. Even so, there are numerous directions for interesting future work. We briefly review some of these directions here.

Including context information. In almost all current work on trajectory analysis it is assumed that the entities that generate the trajectories move in an unbounded homogeneous infinite space (i.e. $\mathbb{R}^d$, for some dimension d). No-table exceptions are the work of Buchin, Dodge, and Speckmann [30] and our own work from Chapter 6. In reality, the entities move in a much more complicated space containing roads, fields, lakes, forests, etc. This may influence our analysis. For example, in a forest the entities may need to be closer together to be considered a group than in a field. Ideally, our algorithms should incorporate such contextual information.

Non-point entities. We represented each moving entity by a point. This is again true for virtually all work on trajectory analysis. If our entities are birds that fly in an area of several hundred square kilometers then this assumption is reasonable. However, if the entities are people moving in a small indoor space like a metro station, or hurricanes crossing a county, we may want a more detailed representation. Instead of representing a moving entity by a single point, we can use a (connected) set of points such as a disk or a polygon. We can revisit all our problems in this setting. It is conceivable that some of our methods can be extended in case an entity is a disk. However, in case an entity is represented by a polygon the orientation of these polygons should also be taken into account. This changes the problems significantly, since a trajectory is no longer just a polygonal line in $\mathbb{R}^d \times \mathbb{T}$. Similar problems occur in the in the related field of motion planing, where non-point entities like disks or polygons are more common. Tools from that field like the configuration space may be helpful in this case too [82].

We note that there are situations in which we may want to generalize the problem even further, and allow the representation of the entity to change over time. For example, we can model the movement of a glacier or a coast line by a moving polygon whose shape changes over time.

Algorithms for massive data. A massive amount of trajectory data has already been captured. The popular sports-tracking website strava.com reported that as of November 2014 their users had uploaded a total of 160 million trajectories, containing a total of 375 billion vertices [109]. This accounts to at least eight terrabyte of data, hence such data sets no longer fit in main memory of a computer. As a result, the running time of our algorithms is no longer just determined by the number of operations executed by the CPU. Instead, the running time is largely determined by the time it takes to transfer data between main memory and disk during the computation.

To perform analysis tasks on such large amounts of (trajectory) data, we need to redesign our algorithms, and analyze them in a computation model that incorporates the number of memory transfers. The two most common such models are the IO model and the cache oblivious model [3, 55]. There is virtually no existing work on trajectory analysis in these models yet.

Online and distributed algorithms. For all tasks considered, we assumed that the trajectories of all moving entities have been collected before, and given to us as a whole. However, we can also consider the situation in which we are given the trajectories while the moving entities are generating them, and we wish to perform the analysis task in real time. This introduces additional complications. Consider for example our central trajectories problem: our central trajectory corresponded to a shortest path in the weighted Reeb graph $\mathcal{R}$. As the entities move, this graph changes, and thus the shortest path may change. By adding a single edge to $\mathcal{R}$, we may suddenly get a completely different different representation from the one computed so far. This may not be desirable. To deal with issues like these we can use online algorithms and competitive analysis [52].

In the problem sketched above, we still assumed that all required information was somehow transmitted to a central computer running our algorithm. However, even that may not be the case. We may be in a situation in which there is no central computer, instead all computation has to happen on the devices carried by the moving entities. For example, the device tracking the movement of a bird may want to know if the bird is currently in a group, so that it can enable an extra sensor and take additional measurements. In such a scenario we need distributed algorithms. Some initial work has recently appeared in this direction [21, 23].

Movement models. We defined a trajectory as a piecewise linear function mapping time to a point in space. Thus, we assumed that in between the vertices of the trajectory, the entity moved along a line with constant speed. We may wish to generalize this to a larger class of movements. In contrast, in many kinetic data structures it is sufficient if the movement of the entities can be described by pseudo-algebraic curves of low degree [13].

Instead of considering a more general movement model, we may also consider a more restricted model. Our lower bounds for the tasks involving multiple entities involve fairly contrived constructions that are highly unlikely to occur in practice. So, we may want to disregard such input trajectories for the efficiency analysis. This can be done using realistic input assumptions. Realistic input assumptions are common in problems involving terrains. For example, one can construct (a triangular irregular network (TIN)) representing a terrain on which the visibility of a single view point has quadratic complexity. By assuming that the triangles of the terrain are not too steep and not too skinny, the complexity can be reduced to only $O(n\sqrt{n})$, where n the number of vertices in the TIN [96]. For trajectory data this yields an interesting modeling question: what constitutes realistic input? This may be a hard question however, considering that for most of the problems we considered the complexity is high even if the entities are just moving along lines. Indeed, in practically all our lower bounds we use a construction where the entities move along lines, and repeat this same construction until the trajectories have their desired length.

Practical tools. We have taken an theoretical point of view on trajectory analysis. We have considered the analysis tasks as a source of interesting “puzzles” for us to solve. But let us not forget the practitioners: the people that have the trajectory data, and wish to analyze it. Even though we solved the four problems (tasks) that we considered, there is still a long way to go before our solutions make it to the hands of the practitioners. The only algorithm from this thesis that has been implemented is the one to compute the trajectory grouping structure, and that implementation should be regarded only as a proof of concept. Implementing our algorithms, and tools to make them usable to practitioners, is still an important remaining task.

Samenvatting

Technologische ontwikkelingen zoals het Global Positioning System (GPS) hebben het mogelijk gemaakt om gemakkelijk objecten, personen of dieren te volgen. Het resultaat is een enorme collectie trajectory-data. Een *trajectory* beschrijft de beweging van een object, persoon of dier[1] gedurende een tijdspanne. Een trajectory geeft dus de locatie van het object als een functie van de tijd. Trajectory-data bevat een schat van informatie, maar het analyseren van trajectory-data is een lastig en tijdrovend proces. Een proces dat we graag zouden automatiseren. Dat is het hoofddoel in dit proefschrift. We bekijken een aantal analysetaken en ontwikkelen algoritmen en technieken om deze taken door de computer uit te laten voeren. We zijn met name geïnteresseerd in de technieken en algoritmen zelf en niet zozeer in het resultaat van de analyse op een gegeven invoer-trajectory, of verzameling van invoer-trajectories.

We bekijken vier analysetaken. Voor elke taak gebruiken we een aanpak kenmerkend voor de computationele meetkunde: we formaliseren de analysetaak en onderzoeken de geometrische eigenschappen die een rol spelen in het probleem. We gebruiken deze eigenschappen om efficiënte (snelle) en gegarandeerd correcte algoritmen te ontwerpen. In veel gevallen vereist dit het gebruik en ontwerp van interessante technieken.

De analysetaken die we beschouwen zijn:

- Het *segmenteren* van een invoer-trajectory op basis van een gegeven niet-monotoon criterium. Bij het segmenteren van een trajectory splitsen we het trajectory in stukken zodanig dat elk stuk (*segment*) voldoet aan het gegeven criterium. Bijvoorbeeld, segmenteer het trajectory zodanig dat op elk segment het verschil tussen de minimum- en maximum-snelheid ten hoogste 20 kilometer per uur is. Als een criterium dat op een segment S voldaan is ook voldaan is op elk sub-segment van S dan is het criterium monotoon. Voor de klasse van monotone criteria is een efficiënt algoritme bekend. We onderzoeken het geval dat het criterium niet-monotoon is en ontwikkelen efficiënte algoritmen voor verschillende zulke criteria.

---

[1] In het vervolg gebruiken we enkel nog de term "object" om te verwijzen naar datgene dat beweegt en dus het trajectory produceert.

- Het vinden van *hotspots*: plekken waar het bewegende object een groot deel van zijn tijd doorbrengt. Er zijn verschillende versies van dit probleem, bijvoorbeeld, wat is de locatie van de kleinste hotspot waarbinnen het object tenminste een uur verblijft. We kunnen ook de grootte van de hotspot vast leggen en vragen om de locatie van een hotspot (van de gegeven grootte) waarbinnen het object de meeste tijd verblijft. We beschouwen zes dergelijke varianten en beschrijven voor elke variant een efficiënt algoritme. Onze algoritmen zijn bovendien makkelijk uit te breiden naar het geval waarin we meerdere invoer-trajectories hebben.

- Het berekenen van alle groepen en hun onderlinge relaties. Een *groep* is een verzameling van gezamenlijk bewegende objecten. We zijn enkel geïnteresseerd in relevante groepen: groepen die groot genoeg zijn en waarin de objecten lang genoeg dicht genoeg bij elkaar in de buurt blijven. Daarnaast, zijn we geïnteresseerd in de relaties tussen de (relevante) groepen. Een grote groep ontstaat wanneer twee kleinere groepen samenkomen en stopt te bestaan wanneer de objecten in kleinere groepen opsplitsen. We vatten al deze informatie samen in wat we de *trajectory grouping structure* noemen.

  Het is gebruikelijk in trajectory-analyse om aan te nemen dat de objecten die de trajectories genereren bewegen in oneindige "lege" ruimte (i.e. $\mathbb{R}^d$). Dat is duidelijk geen realistische aanname. In werkelijkheid hebben de objecten te maken met allerlei obstakels zoals gebouwen en rivieren. Voor het berekenen van de trajectory grouping structure beschouwen we ook dit scenario en analyseren we hoe de looptijd van onze algoritmen af hangt van het toegestane type obstakels.

- Het vinden van een *representative trajectory* voor een gegeven verzameling aan invoer-trajectories. Een representative trajectory dient de gezamelijke kenmerken van de invoer-trajectories samen te vatten. We geven een definitie voor een representative trajectory, een *central trajectory*, dat zowel de tijd als de locatie van de objecten in acht neemt. Een central trajectory bestaat uit stukken van de invoer-trajectories en is ten alle tijden "zo centraal als mogelijk". We formaliseren dit en presenteren efficiënt algoritmen om een central trajectory te berekenen, zelfs wanneer de objecten bewegen in $\mathbb{R}^d$, voor een arbitraire (constante) dimensie d. Onze resultaten voor het berekenen van de trajectory grouping structure spelen ook hier een belangrijke rol.

Bibliography

[1] P. K. Agarwal, M. de Berg, J. Gao, L. J. Guibas, and S. Har-Peled. "Staying in the Middle: Exact and Approximate Medians in $\mathbb{R}^1$ and $\mathbb{R}^2$ for Moving Points." In: *CCCG*. 2005, pp. 43–46.

[2] P. K. Agarwal and M. Sharir. "Davenport-Schinzel Sequences and Their Geometric Applications." In: *Handbook of Computational Geometry*. Ed. by J.-R. Sack and J. Urrutia. Amsterdam: Elsevier Science Publishers B.V. North-Holland, 2000, pp. 1–47.

[3] A. Aggarwal, J. Vitter, et al. "The input/output complexity of sorting and related problems." In: *Communications of the ACM* 31.9 (1988), pp. 1116–1127.

[4] S. P. Alewijnse, K. Buchin, M. Buchin, S. Sijben, and M. A. Westenberg. "Model-based Segmentation and Classification of Trajectories." In: *Abstracts of the 30th European Workshop on Computational Geometry (EuroCG)* (2014).

[5] H. Alt, B. Behrends, and J. Blömer. "Approximate Matching of Polygonal Shapes." In: *Annals of Mathematics and Artificial Intelligence* 13.3-4 (1995), pp. 251–265.

[6] H. Alt and M. Godau. "Computing the Fréchet distance between two polygonal curves." In: *International Journal of Computational Geometry & Applications* 5 (1995), pp. 75–91.

[7] L. O. Alvares, V. Bogorny, B. Kuijpers, J. de Macêdo, B. Moelans, and A. Vaisman. "A model for enriching trajectories with semantic geographical information." In: *Proc. 15th ACM International Symposium on Geographic Information Systems*. ACM, 2007, p. 22.

[8] N. M. Amato, M. T. Goodrich, and E. A. Ramos. "Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling." In: *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms*. Vol. 9. 11. 2000, pp. 705–706.

[9] A. Anagnostopoulos, M. Vlachos, M. Hadjieleftheriou, E. Keogh, and P. Yu. "Global distance-based segmentation of trajectories." In: *Proc. 12th Conference on Knowledge Discovery and Data Mining*. 2006, pp. 34–43.

[10] R. Anderson, P. Beanie, and E. Brisson. "Parallel algorithms for arrangements." In: *Algorithmica* 15.2 (1996), pp. 104–125.

[11] G. Andrienko, N. Andrienko, and S. Wrobel. "Visual Analytics Tools for Analysis of Movement Data." In: *SIGKDD Explorrations Newsletter* 9.2 (Dec. 2007), pp. 38–46.

[12] B. Aronov, A. Driemel, M. van Kreveld, M. Löffler, and F. Staals. "Segmentation of Trajectories on Non-Monotone Criteria." In: *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2013, pp. 1897–1911.

[13] J. Basch, L. J. Guibas, and J. Hershberger. "Data Structures for Mobile Data." In: *Journal of Algorithms* 31.1 (1999), pp. 1–28.

[14] R. Basu, B. Bhattacharya, and T. Talukdar. "The Projection Median of a Set of Points in $\mathbb{R}^d$." In: *Discrete & Computational Geometry* 47.2 (2012), pp. 329–346.

[15] M. A. Bender and M. Farach-Colton. "The LCA Problem Revisited." In: *LATIN 2000: Theoretical Informatics*. Ed. by G. Gonnet and A. Viola. Vol. 1776. LNCS. Springer, 2000, pp. 88–94.

[16] M. Benkert, B. Djordjevic, J. Gudmundsson, and T. Wolle. "Finding Popular Places." In: *International Journal of Computational Geometry & Applications* 20.1 (2010), pp. 19–42.

[17] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. "Reporting flock patterns." In: *Computational Geometry* 41.3 (2008), pp. 111–125.

[18] M. Bern, D. Eppstein, P. Plassman, and F. Yao. "Horizon theorems for lines and polygons." In: *Discrete and Computational Geometry: Papers from the DIMACS Special Year*. Ed. by J. Goodman, R. Pollack, and W. Steiger. Vol. 6. DIMACS. Providence, RI: American Mathematical Society, ACM, 1991, pp. 45–66.

[19] D. J. Berndt and J. Clifford. "Using Dynamic Time Warping to Find Patterns in Time Series." In: *KDD workshop*. Vol. 10. 16. 1994, pp. 359–370.

[20] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. "Reeb graphs for shape analysis and applications." In: *Theoretical Computer Science* 392.1-3 (2008), pp. 5–22.

[21] S. Bleisch, M. Duckham, A. Galton, P. Laube, and J. Lyon. "Mining candidate causal relationships in movement patterns." In: *International Journal of Geographical Information Science* 28.2 (2014), pp. 363–382.

[22] M. Bodlaender, C. Hurkens, V. Kusters, F. Staals, G. Woeginger, and H. Zantema. "Cinderella versus the Wicked Stepmother." In: *Theoretical Computer Science*. Ed. by J. Baeten, T. Ball, and F. Boer. Vol. 7604. LNCS. Springer, 2012, pp. 57–71.

[23] A. Both, M. Duckham, P. Laube, T. Wark, and J. Yeoman. "Decentralized Monitoring of Moving Objects in a Transportation Network Augmented with Checkpoints." In: *The Computer Journal* 56.12 (2013), pp. 1431–1449.

[24] P. Bovet and S. Benhamou. "Spatial analysis of animals' movements using a correlated random walk model." In: *Journal of Theoretical Biology* 131.4 (1988), pp. 419–433.

[25] K. Buchin, M. Buchin, and J. Gudmundsson. "Detecting Single File Movement." In: *Proc. 16th ACM International Conference on Advances in Geographic Information Systems*. 2008, pp. 288–297.

[26] K. Buchin, M. Buchin, M. van Kreveld, M. Löffler, R. I. Silveira, C. Wenk, and L. Wiratma. "Median Trajectories." In: *Algorithmica* 66.3 (2013), pp. 595–614.

[27] K. Buchin, M. Buchin, M. van Kreveld, and J. Luo. "Finding long and similar parts of trajectories." In: *Computational Geometry* 44.9 (2011), pp. 465–476.

[28] K. Buchin, M. Buchin, M. van Kreveld, B. Speckmann, and F. Staals. "Trajectory Grouping Structure." In: *Journal of Computational Geometry* 6.1 (2015), pp. 75–98.

[29] K. Buchin, V. Kusters, B. Speckmann, F. Staals, and B. Vasilescu. "A splitting line model for directional relations." In: *Proc. 19th International Conference on Advances in Geographic Information Systems*. GIS '11. Chicago, Illinois: ACM, 2011, pp. 142–151.

[30] M. Buchin, S. Dodge, and B. Speckmann. "Context-Aware Similarity of Trajectories." In: *Geographic Information Science*. Vol. 7478. LNCS. Springer, 2012, pp. 43–56.

[31] M. Buchin, A. Driemel, M. J. van Kreveld, and V. Sacristan. "Segmenting trajectories: A framework and algorithms using spatiotemporal criteria." In: *Journal of Spatial Information Science* 3.1 (2011), pp. 33–63.

[32] C. Calenge, S. Dray, and M. Royer-Carenzi. "The concept of animals' trajectories from a data analysis perspective." In: *Ecological Informatics* 4.1 (2009), pp. 34–41.

[33] F. Chazal and J. Sun. "Gromov-Hausdorff Approximation of Filament Structure Using Reeb-type Graph." In: *Proc. 30th Annual Symposium on Computational Geometry*. SOCG'14. Kyoto, Japan: ACM, 2014, pp. 491–500.

[34] B. Chazelle. "Triangulating a simple polygon in linear time." In: *Discrete Computational Geometry* 6.5 (1991), pp. 485–524.

[35] P. Chundi and D. J. Rosenkrantz. "Segmentation of Time Series Data." In: *Encyclopedia of Data Warehousing and Mining*. Ed. by J. Wang. 2009, pp. 1753–1758.

[36] S. Dasgupta, C. Papadimitriou, and U. Vazirani. *Algorithms*. McGraw-Hill, 2008.

[37] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. 2nd. Berlin, Germany: Springer-Verlag, 2000.

[38] E. D. Demaine, S. Eisenstat, L. J. Guibas, and A. Schulz. "Kinetic minimum spanning circle." In: *Proc. of the Fall Workshop on Computational Geometry*. 2010.

[39] T. K. Dey and Y. Wang. "Reeb graphs: approximation and persistence." In: *Proc. 27th ACM Symposium on Computational Geometry*. SoCG '11. 2011, pp. 226–235.

[40] S. Dodge, R. Weibel, and E. Forootan. "Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects." In: *Computers, Environment and Urban Systems* 33.6 (2009), pp. 419–434.

[41] S. Dodge, R. Weibel, and A.-K. Lautenschütz. "Towards a taxonomy of movement patterns." In: *Information Visualization* 7.3-4 (2008), pp. 240–252.

[42] D. H. Douglas and T. K. Peucker. "Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature." In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2 (1911), pp. 112–122.

[43] A. Driemel, S. Har-Peled, and C. Wenk. "Approximating the Fréchet Distance for Realistic Curves in Near Linear Time." In: *Discrete & Computational Geometry* 48.1 (2012), pp. 94–127.

[44] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. "Making data structures persistent." In: *Journal of Computer and System Sciences* 38.1 (1989), pp. 86–124.

[45] S. Durocher and D. Kirkpatrick. "The projection median of a set of points." In: *Computational Geometry* 42.5 (2009), pp. 364–375.

[46] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. "Optimal point location in a monotone subdivision." In: *SIAM Journal on Computuing* 15.2 (1986), pp. 317–340.

[47] H. Edelsbrunner and J. L. Harer. *Computational Topology – an introduction*. American Mathematical Society, 2010.

[48] H. Edelsbrunner, J. Harer, A. Mascarenhas, V. Pascucci, and J. Snoeyink. "Time-varying Reeb graphs for continuous space-time data." In: *Computational Geometry* 41.3 (2008), pp. 149–166.

[49] H. Edelsbrunner, D. Letscher, and A. Zomorodian. "Topological Persistence and Simplification." In: *Discrete & Computational Geometry* 28 (4 2002), pp. 511–533.

[50] D. Eppstein, M. van Kreveld, B. Speckmann, and F. Staals. "Improved Grid Map Layout by Point Set Matching." In: *IEEE PacificVis*. IEEE, 2013, pp. 25–32.

[51] M. Ester, H. Kriegel, J. Sander, and X. Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *Proc. 2nd International Conference on Knowledge Discovery and Data mining*. Vol. 1996. AAAI Press. 1996, pp. 226–231.

[52] W. Evans, D. Kirkpatrick, M. Löffler, and F. Staals. "Competitive Query Strategies for Minimising the Ply of the Potential Locations of Moving Points." In: *Proc. 29th Annual Symposium on Computational Geometry*. SoCG '13. Rio de Janeiro, Brazil: ACM, 2013, pp. 155–164.

[53] P. Fauchald and T. Tveraa. "Using First-Passage Time in the Analysis of Area-Restricted Search and Habitat Selection." In: *Ecology* 84.2 (2003), pp. 282–288.

[54] A. Fomenko and T. Kunii, eds. *Topological Methods for Visualization*. Tokyo, Japan: Springer, 1997.

[55] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. "Cache-oblivious algorithms." In: *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE. 1999, pp. 285–297.

[56] A. Fujimura and K. Sugihara. "Geometric analysis and quantitative evaluation of sport teamwork." In: *Systems and Computers in Japan* 36.6 (2005), pp. 49–58.

[57] S. Gaffney and P. Smyth. "Trajectory Clustering with Mixtures of Regression Models." In: *Proc. 5th International Conference on Data Discovery and Data Mining*. 1999, pp. 63–72.

[58] X. Ge, I. Safa, M. Belkin, and Y. Wang. "Data Skeletonization via Reeb Graphs." In: *Proc. 25th Annual Conference on Neural Information Processing Systems*. NIPS '11. 2011, pp. 837–845.

[59] J. Gudmundsson, P. Laube, and T. Wolle. "Movement Patterns in Spatio-Temporal Data." In: *Encyclopedia of GIS*. Ed. by S. Shekhar and H. Xiong. Springer, 2008, pp. 726–732.

[60] J. Gudmundsson and M. van Kreveld. "Computing longest duration flocks in trajectory data." In: *Proc. 14th ACM International Symposium on Advances in Geographic Information Systems*. SIGSPATIAL '06. ACM, 2006, pp. 35–42.

[61] J. Gudmundsson, M. van Kreveld, and B. Speckmann. "Efficient Detection of Patterns in 2D Trajectories of Moving Points." In: *GeoInformatica* 11 (2 2007), pp. 195–215.

[62] J. Gudmundsson, M. van Kreveld, and B. Speckmann. "Efficient detection of patterns in 2D trajectories of moving points." In: *GeoInformatica* 11 (2007), pp. 195–215.

[63] J. Gudmundsson, M. van Kreveld, and F. Staals. "Algorithms for Hotspot Computation on Trajectory Data." In: *Proc. 21th International Conference on Advances in Geographic Information Systems*. SIGSPATIAL '13. Orlando, Florida: ACM, 2013, pp. 134–143.

[64] L. J. Guibas and J. Hershberger. "Optimal shortest path queries in a simple polygon." In: *Journal of Computer and System Sciences* 39.2 (1989), pp. 126–152.

[65] L. J. Guibas, J. Hershberger, J. S. B. Mitchell, and J. Snoeyink. "Approximating Polygons and Subdivisions with Minimum Link Paths." In: *International Journal of Computational Geometry & Applications* 3.4 (1993), pp. 383–415.

[66] E. Gurarie, R. D. Andrews, and K. L. Laidre. "A novel method for identifying behavioural changes in animal movement data." In: *Ecology Letters* 12.5 (2009), pp. 395–408.

[67] S. Har-Peled and B. Raichel. "The Fréchet Distance Revisited and Extended." In: *Proc. 27th Annual Symposium on Computational Geometry*. SoCG '11. Paris, France: ACM, 2011, pp. 448–457.

[68] J. Hershberger and S. Suri. "An Optimal Algorithm for Euclidean Shortest Paths in the Plane." In: *SIAM Journal on Computing* 28.6 (1999), pp. 2215–2256.

[69] J. S. Horne, E. O. Garton, S. M. Krone, and J. S. Lewis. "Analyzing animal movements using Brownian bridges." In: *Ecology* 88.9 (2007), pp. 2354–2363.

[70] Y. Huang, C. Chen, and P. Dong. "Modeling Herds and Their Evolvements from Trajectory Data." In: *Geographic Information Science*. Vol. 5266. LNCS. Springer, 2008, pp. 90–105.

[71] F. Hurtado, M. Löffler, I. Matos, V. Sacristan, M. Saumell, R. Silveira, and F. Staals. "Terrain visibility with multiple viewpoints." In: *Proc. 24th International Symposium on Algorithms and Computation*. LNCS. Hong Kong, Hong Kong: Springer, 2013, pp. 317–327.

[72] S.-Y. Hwang, Y.-H. Liu, J.-K. Chiu, and E.-P. Lim. "Mining Mobile Group Patterns: A Trajectory-Based Approach." In: *Advances in Knowledge Discovery and Data Mining*. Vol. 3518. LNCS. Springer, 2005, pp. 145–146.

[73] H. Imai and M. Iri. "Polygonal approximations of a curve – formulations and algorithms." In: *Computational Morphology*. Elsevier Science, 1988, pp. 71–86.

[74] J. Lee, J. Han, and K.-Y. Whang. "Trajectory clustering: a partition-and-group framework." In: *Proc. ACM International Conference on Management of Data*. 2007, pp. 593–604.

[75] A. Jain, M. N. Murty, and P. Flynn. "Data Clustering: A Review." In: *ACM Computing Surveys* 31.3 (1999), pp. 264–323.

[76] H. Jeung, M. Yiu, and C. Jensen. "Trajectory pattern mining." In: *Computing with Spatial Trajectories*. Ed. by Y. Zheng and X. Zhou. Springer, 2011, pp. 143–177.

[77] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. "Discovery of convoys in trajectory databases." In: *PVLDB* 1 (1 2008), pp. 1068–1080.

[78] A. Johnson, J. Wiens, B. Milne, and T. Crist. "Animal movements and population dynamics in heterogeneous landscapes." In: *Landscape Ecology* 7 (1 1992), pp. 63–75.

[79] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. "Detecting Commuting Patterns by Clustering Subtrajectories." In: *International Journal Computational Geometry & Applications* 21.3 (2011), pp. 253–282.

[80] P. Kalnis, N. Mamoulis, and S. Bakiras. "On Discovering Moving Clusters in Spatio-temporal Data." In: *Advances in Spatial and Temporal Databases*. Vol. 3633. LNCS. Springer, 2005, pp. 364–381.

[81] F. Kammer, M. Löffler, P. Mutser, and F. Staals. "Practical Approaches to Partially Guarding a Polyhedral Terrain." In: *Proc. 8th International Conference on Geographic Information Science*. Vol. 8728. LNCS. Vienna, Austria: Springer, 2014, pp. 318–332.

[82] K. Kedem, R. Livne, J. Pach, and M. Sharir. "On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles." In: *Discrete & Computational Geometry* 1.1 (1986), pp. 59–71.

[83] E. J. Keogh. "Exact Indexing of Dynamic Time Warping." In: *Proc. 28th International Conference on Very Large Data Bases*. 2002, pp. 406–417.

[84] I. Kostitsyna, M. van Kreveld, M. Löffler, B. Speckmann, and F. Staals. "Trajectory Grouping Structure under Geodesic Distance." In: *Proc. 31th annual Symposium on Computational Geometry*. SoCG '15. Eindhoven, The Netherlands: Lipics, 2015.

[85] P. Laube, M. van Kreveld, and S. Imfeld. "Finding REMO – detecting relative motion patterns in geospatial lifelines." In: *Developments in Spatial Data Handling: 11th International Symposium on Spatial Data Handling*. Ed. by P. Fisher. 2004, pp. 201–215.

[86] J.-G. Lee, J. Han, X. Li, and H. Gonzalez. "*TraClass*: Trajectory Classification using Hierarchical Region-based and Trajectory-based Clustering." In: *PVLDB '08*. 2008, pp. 1081–1094.

[87] X. Li, X. Li, D. Tang, and X. Xu. "Deriving features of traffic flow around an intersection from trajectories of vehicles." In: *Proc. 18th International Conference on Geoinformatics*. IEEE. 2010, pp. 1–5.

[88] Z. Li, B. Ding, J. Han, and R. Kays. "Swarm: Mining Relaxed Temporal Moving Object Clusters." In: *PVLDB* 3.1 (2010), pp. 723–734.

[89] M. Löffler, M. Kaiser, T. van Kapel, G. Klappe, M. van Kreveld, and F. Staals. "The Connect-The-Dots Family of Puzzles: Design and Automatic Generation." In: *ACM Transactions on Graphics* 33.4 (July 2014). This work was presented at SigGraph 2014., 72:1–72:10.

[90] M. Löffler, M. Nöllenburg, and F. Staals. "Mixed Map Labeling." In: *Proc. 9th International Conference on Algorithms and Complexity*. LNCS. Paris, France: Springer, 2015.

[91] M. Nanni and D. Pedreschi. "Time-focused clustering of trajectories of moving objects." In: *Journal of Intelligent Information Systems* 27 (2006), pp. 285–289.

[92] M. Vlachos, D. Gunopulos, and G. Kollios. "Discovering similar multidimensional trajectories." In: *Proc. 18th International Conference on Data Engineering*. 2002, pp. 673–684.

[93] R. Mann, A. Jepson, and T. El-Maraghi. "Trajectory segmentation using dynamic programming." In: *Proc. 16th International Conference on Pattern Recognition (ICPR)*. Vol. 1. 2002, pp. 331–334.

[94] N. Megiddo. "Applying parallel computation algorithms in the design of serial algorithms." In: *Journal of the ACM* 30.4 (1983), pp. 852–865.

[95] H. J. Miller. "Modelling accessibility using space-time prism concepts within geographical information systems." In: *International journal of geographical information systems* 5.3 (1991), pp. 287–301.

[96] E. Moet, M. van Kreveld, and A. F. van der Stappen. "On Realistic Terrains." In: *Proc. 22nd Annual Symposium on Computational Geometry*. SoCG '06. Sedona, Arizona, USA: ACM, 2006, pp. 177–186.

[97] B. Moreno, V. Times, C. Renso, and V. Bogorny. "Looking Inside the Stops of Trajectories of Moving Objects." In: *Proc. XI Brazilian Symposium on Geoinformatics*. MCT/INPE, 2010, pp. 9–20.

[98] D. Mount, R. Silverman, and A. Wu. "On the Area of Overlap of Translated Polygons." In: *Computer Vision and Image Understanding* 64.1 (1996), pp. 53–61.

[99] R. Nathan, W. Getz, E. Revilla, M. Holyoak, R. Kadmon, D. Saltz, and P. Smouse. "A movement ecology paradigm for unifying organismal movement research." In: *Proc. National Academy of Sciences* 105 (2008), pp. 19052–19059.

[100] Oregon Department of Fish and Wildlife and the USDA Forest Service. *The Starkey Project*. 2004.

[101] J. Pach and P. K. Agarwal. *Combinatorial Geometry*. New York, NY: John Wiley & Sons, 1995.

[102] A. Palma, V. Bogorny, B. Kuijpers, and L. O. Alvares. "A clustering-based approach for discovering interesting places in trajectories." In: *Proc. 2008 ACM Symposium on Applied Computing*. 2008, pp. 863–868.

[103] S. Parsa. "A deterministic $O(m \log m)$ time algorithm for the Reeb graph." In: *Proc. 28th ACM Symposium on Computational Geometry*. 2012, pp. 269–276.

[104] Y. Shiloach and U. Vishkin. "An $O(\log n)$ parallel connectivity algorithm." In: *Journal of Algorithms* 3.1 (1982), pp. 57–67.

[105] B. Shneiderman. "The eyes have it: a task by data type taxonomy for information visualizations." In: (1996), pp. 336–343.

[106] D. D. Sleator and R. E. Tarjan. "A data structure for dynamic trees." In: *Journal of Computer and System Sciences* 26.3 (1983), pp. 362–391.

[107] S. Spaccapietra, C. Parent, M. Damiani, J. de Macêdo, F. Porto, and C. Vangenot. "A conceptual view on trajectories." In: *Data & Knowledge Engineering* 65.1 (2008), pp. 126–146.

[108] A. Stohl. "Computation, accuracy and applications of trajectories – A review and bibliography." In: *Atmospheric Environment* 32.6 (1998), pp. 947–966.

[109] Strava. *Strava.com*. Feb. 3, 2015. URL: `http://engineering.strava.com/global-heatmap/`.

[110] D. Sumpter. *Collective Animal Behavior*. Princeton: Princeton University Press, 2010.

[111] R. E. Tarjan and U. Vishkin. "An efficient parallel biconnectivity algorithm." In: *SIAM Journal on Computing* 14.4 (1985), pp. 862–874.

[112] E. Terzi and P. Tsaparas. "Efficient Algorithms for Sequence Segmentation." In: *Proc. 6th SIAM International Conference on Data Mining*. 2006, pp. 314–325.

[113] S. Tiwari and S. Kaushik. "Mining Popular Places in a Geo-spatial Region Based on GPS Data Using Semantic Information." In: *Proc. 8th International Workshop on Databases in Networked Information Systems*. Vol. 7813. LNCS. Springer, 2013, pp. 262–276.

[114] M. van Kreveld, M. Löffler, and F. Staals. "Central Trajectories." In: *CoRR* abs/1501.01822 (2015).

[115] B. van Moorter, D. R. Visscher, C. L. Jerde, J. L. Frair, and E. H. Merrill. “Identifying Movement States From Location Data Using Cluster Analysis.” In: *J. Wildlife Management* 74.3 (2010), pp. 588–594.

[116] F. Verhein and S. Chawla. “Mining Spatio-temporal Association Rules, Sources, Sinks, Stationary Regions and Thoroughfares in Object Mobility Databases.” In: *Proc. 11th International Conference on Database Systems for Advanced Applications*. Vol. 3882. LNCS. Springer, 2006, pp. 187–201.

[117] M. R. Vieira, P. Bakalov, and V. J. Tsotras. “On-line discovery of flock patterns in spatio-temporal data.” In: *Proc. 17th ACM International Conference on Advances in Geographic Information Systems*. SIGSPATIAL ’09. ACM, 2009, pp. 286–295.

[118] Y. Wang, E.-P. Lim, and S.-Y. Hwang. “Efficient algorithms for mining maximal valid groups.” In: *The VLDB Journal* 17.3 (May 2008), pp. 515–535.

[119] U. Wilensky. *NetLogo*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. 1999.

[120] U. Wilensky. *NetLogo Flocking model*. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. 1998.

[121] H. Yoon and C. Shahabi. “Robust Time-Referenced Segmentation of Moving Object Trajectories.” In: *Proc. 8th IEEE International Conference on Data Mining*. 2008, pp. 1121–1126.

Curriculum Vitae

Frank Staals was born on 10 August 1988 in Eindhoven. He finished his pre-university education in 2006 at the SG Were Di in Valkenswaard. In 2009 he received his Bachelor degree in Computer Science and Engineering from the Eindhoven University of Technology. He then continued with a Master in Computer Science and Engineering, also at the Eindhoven University of Technology. During this period he was a student in the honors program. His Master thesis was on geographic grid embeddings, under the supervision of prof. dr. Bettina Speckmann. He received his Master of Science degree (*cum laude*) in August 2011. As of September 2011 he has been working as a PhD Student in the Multimedia & Geometry, and in the Virtual Words group at Utrecht University. The results of his PhD are presented in this thesis.

Other Publications

In addition to the four publications incorporated into this thesis, Frank has published the following formally reviewed articles.

[90] M. Löffler, M. Nöllenburg, and F. Staals. "Mixed Map Labeling." In: *Proc. 9th International Conference on Algorithms and Complexity*. LNCS. Paris, France: Springer, 2015

[81] F. Kammer, M. Löffler, P. Mutser, and F. Staals. "Practical Approaches to Partially Guarding a Polyhedral Terrain." In: *Proc. 8th International Conference on Geographic Information Science*. Vol. 8728. LNCS. Vienna, Austria: Springer, 2014, pp. 318–332

[89] M. Löffler, M. Kaiser, T. van Kapel, G. Klappe, M. van Kreveld, and F. Staals. "The Connect-The-Dots Family of Puzzles: Design and Automatic Generation." In: *ACM Transactions on Graphics* 33.4 (July 2014). This work was presented at SigGraph 2014., 72:1–72:10

[71] F. Hurtado, M. Löffler, I. Matos, V. Sacristan, M. Saumell, R. Silveira, and F. Staals. "Terrain visibility with multiple viewpoints." In: *Proc. 24th International Symposium on Algorithms and Computation*. LNCS. Hong Kong, Hong Kong: Springer, 2013, pp. 317–327

[52] W. Evans, D. Kirkpatrick, M. Löffler, and F. Staals. "Competitive Query Strategies for Minimising the Ply of the Potential Locations of Moving

Points." In: *Proc. 29th Annual Symposium on Computational Geometry*. SoCG '13. Rio de Janeiro, Brazil: ACM, 2013, pp. 155–164

[50] D. Eppstein, M. van Kreveld, B. Speckmann, and F. Staals. "Improved Grid Map Layout by Point Set Matching." In: *IEEE PacificVis*. IEEE, 2013, pp. 25–32

[22] M. Bodlaender, C. Hurkens, V. Kusters, F. Staals, G. Woeginger, and H. Zantema. "Cinderella versus the Wicked Stepmother." In: *Theoretical Computer Science*. Ed. by J. Baeten, T. Ball, and F. Boer. Vol. 7604. LNCS. Springer, 2012, pp. 57–71

[29] K. Buchin, V. Kusters, B. Speckmann, F. Staals, and B. Vasilescu. "A splitting line model for directional relations." In: *Proc. 19th International Conference on Advances in Geographic Information Systems*. GIS '11. Chicago, Illinois: ACM, 2011, pp. 142–151

Acknowledgments

The approximately 170 pages in this thesis present some of my work from the last four years. However, without the help and support of various others, this thesis would not have existed, let alone have been as much fun to work on as it has been. This section is my attempt to thank those people. My apologies to anyone I may forget.

To start, I should actually thank the people that made me want to do a PhD in the first place. In particular, Dennis Schunselaar, Vincent Kusters, and Bogdan Vasilescu. I have always enjoyed studying with you guys, and the friendly "competition" of who had the best grades. You made me strive for excellence, and got me interested in research. Also, a big thanks to Bettina Speckmann and Kevin Buchin for introducing me to the wonderful world of geometric algorithms during the Honors program and my Master project.

As for during my PhD itself, it was a huge privilege and a great joy to work with my supervisors Marc van Kreveld and Maarten Löffler. I am still impressed by all their knowledge of geometric algorithms, their ability to explain their ideas in a clear and concise manner, and how they manage to extract the core from my rambling explanations. They are also incredibly friendly and helpful. I could always walk into Marc's office with questions, and I really enjoyed the discussions with Maarten while drinking a "dark-colored hot beverage".[2] Above all, I have always loved the fact that I could do research *with* them rather than for them, and I am very grateful for everything that they taught me throughout these years.

I would also like to thank all my other co-authors. Not only those with whom I worked on the topics presented in this thesis (Boris Aronov, Kevin Buchin, Maike Buchin, Anne Driemel, Joachim Gudmundsson, Irina Kostitsyna, and Bettina Speckmann), but also those with whom I worked on many other interesting problems (Marijke Bodlaender, David Eppstein, Will Evans, Cor Hurkens, Ferran Hurtado, Mira Kaiser, Frank Kammer, Tim van Kapel, David Kirkpatrick, Gerwin Klappe, Vincent Kusters, Inês Matos, Paul Mutser, Martin Nöllenburg, Vera Sacristan, Maria Saumell, Rodrigo Silveira, Bogdan Vasilescu, Gerhard Woeginger, and Hans Zantema). It was wonderful working with all of you. Additionally, I would like to the to thank the members of my reading

---

[2] which, we concluded, was a good common indicator for coffee and tea.

committee (Mark de Berg, Hans Bodlaender, Harry Buhrman, Rodrigo Silveira, and Jack Snoeyink), for reading my entire thesis, and for their useful comments.

Working in Utrecht was a joy, not just because of Marc and Maarten, but also thanks to my other colleagues from the Interaction Technology group and the Virtual Worlds group. I really enjoyed the lunch-time discussions with Bas, Geert-Jan, Frans, Anja, and all the others about all sorts of things.

Throughout the last four years I have had the chance to attend conferences and workshops all around the world. On most of these trips I was accompanied by others from Utrecht or Eindhoven. To my travel companions: it was great fun traveling, and sightseeing with all of you. At every conference and workshop I got the chance to meet new interesting people, and meet up with people that I encountered before. I have had so many interesting discussions, all of which made my life as a scientist incredibly fun and interesting.

Luckily, also I still had some pass time during my PhD. A large part of that time I spent mountain biking, often in the company of friends. I absolutely loved those times, and thus I would like to thank everyone from the racing-group of MBC Midden Nederland and all my other cycling-friends. I hope that we may find many more nice trails to ride in the future. Of course also a big thanks to everyone from TG and all my non-cycling-related friends.

Finally, I would like to thank my family; my sisters Astrid and Karin, but in particular my parents. Pap, mam, als ik hier zou willen opschrijven wat jullie allemaal voor me gedaan hebben dan had ik in een handomdraai nog eens 170 pagina's. Jullie hebben me altijd en in alles gesteund en geholpen. Daarvoor ben ik jullie ontzettend dankbaar. Ik kan me geen betere, of lievere, ouders voorstellen. Bedankt!

- Frank